Learning tasks using bicycle teaching materials

First edition September 25, 2018

Tamaden Industries Co., Ltd.

Chapter 1 About the vehicle body

1.1 Vehicle structure

1.1.1 Caster angle/trail/offset

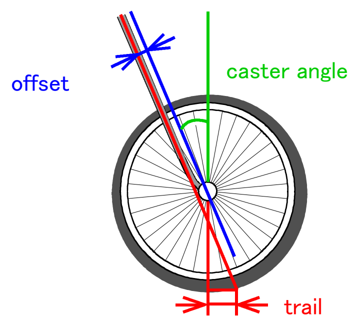


Figure 1.1

The front wheel od the bicycle has caster angle,trail, and There are three elements of offset,and each element is Together,they affect driving performance.Figure 1.1

caster angle : This is the angle of inclination of the front wheel fork. Increasing the caster angle also increases the trail.

trail :The distance between the grounding point of the wheel and the point where the extension line of the handle fork axis intersects with the ground. It is said that the longer the trail, the better the running stability and the more hands-off driving becomes possible.

offset .Distance between handle fork axis and wheel axis. Due to the relationship between the offset and the caster angle, a force is generated that tries to return the handle toward the center.

If you search the internet or literature, you will find many explanations about the three elements of the handle. However, the handlebar of this bicycle is operated electrically, so stability and restorability are not required.
On the other hand, if there is a trail, the moment you move the handle electrically, the handle fork axis will move left and right, causing vibrations in the bike, so this bicycle has a trail distance of zero. In other words, the extension line of the handle fork axis is the grounding point of the front wheel. Trail = 0 is an important item when designing a bicycle whose handlebars are operated electrically.

1.1.2 Wheelbase

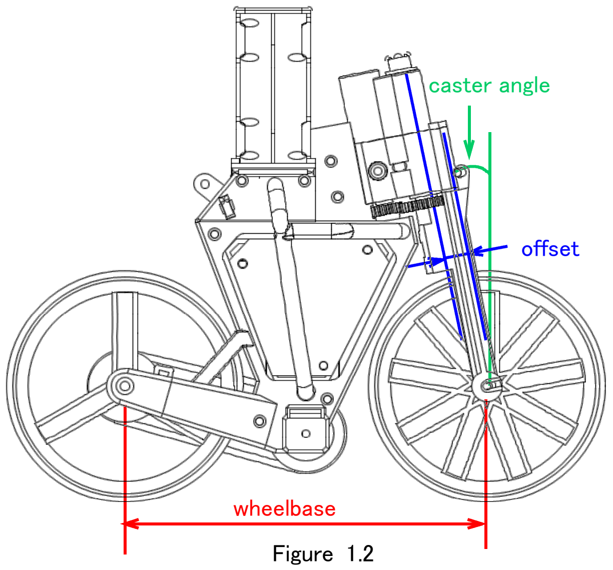


Figure 1.2

The distance between the front and rear axles is called the wheelbase. Figure 1.2
Wheelbase, steering angle and turning angle The relationship is as shown in equation 1.1.

$$\text{turning angle} = K \frac{\text{handle angle}}{\text{Wheelbase}} \quad \cdot\cdot\cdot \text{ Equation 1.1}$$

In other words, the shorter the wheelbase, the better the steering will be.If the wheelbase is lon g, such as a tandem two-seater bicycle, the bicycle will not be able to turn easily. Also,since a long wheelbase results in poor maneuve rability,the speed at which the setting wheel can be moved (steering speed)is also rewuired.

1.1.3 center of gravity

The higher the bicycle's center of gravity is, the longer the natural period of sway will be, resulting in more stable riding.

Natural period of shaking $T=2\pi\sqrt{\frac{h}{g}}$ ...Equation 1.2

h is the height from the ground to the center of gravity

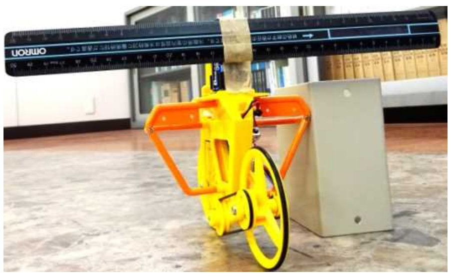
When making a bicycle body by hand, if the wheels and frame of the body are made of material with a high specific gravity, the entire body will feel heavy, and you cannot expect it to ride very well.

summary

The design points of electric bicycles are

.trail. 0 .short wheelbase .center of gravity is high

.Let's try 1.1


photograph 1.1

As shown in Photo 1.1, by attaching a 30cm ruler to the battery case in a Yajirobee shape to raise the center of gravity and giving it the Yajirobee effect, the bicycle's running.characteristics change.
I think it would be interesting to try using weights other than a ruler.
It is also possible to add a flight wheel to this position to control the rotation speed, as in a more advanced version of the ``Murata Seisaku-kun."

The current board only controls the steering wheel and pedals, but it is also possible to prepare a board that can also control a third actuator. (The shape is slightly different.)
Note 1: The metal electrodes are exposed on top of the battery case, so placing conductor weights directly on top of the battery case will cause a short circuit in the battery, which is dangerous.
Please insulate.

1.2 handle motor

1.2.1 Handle motor specifications

photograph 1.2

The handlebars of automatic posture control bicycles are operated by a DC brushed motor (Photo 1-2) with a built-in gear head and incremental encoder manufactured by Citizen Micro Co., Ltd. Motor specifications are shown in Table 1.1, Table 1.2, and Figure 1.3.

Model	IG-10GM-PW1705A-06	Reduction ratio	1/64
Rated voltage	6 〔V〕	Rated current	125 〔mA〕 or less
Output shaft rotation	163±28 〔rpm〕	Output shaft torque	7.35 〔mN･m〕
No-load rotation speed	203±30 〔rpm〕	No load current	90 〔mA〕 or less

Table 1.1 Motor part specifications

Number of pulses	12 [P/R] 12 pulses per motor rotation
maximum response frequency	20KHz
output channel	90° phase A phase/B phase voltage output
power supply	DC3[V]to DC24[V]Current consumption10[mA] or less

Table 1.2 Incremental encoder section specifications

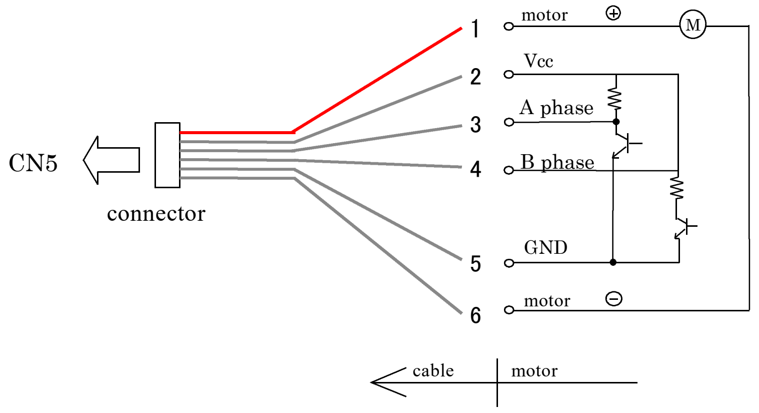


Figure 1.3 Handle motor cable connection

1.2.2 Relationship between handle angle and encoder

The handlebars of automatic posture control bicycles can move approximately 70° left and right, for a total of 140° .

The handle and handle motor are connected with a gear ratio of 20:34, so the rotation angle of the handle angle motor is $(70° + 70°) \times \frac{20}{34} = 238°$

The encoder outputs 12 pulses per motor rotation, so the encoder pulses for a steering wheel angle of 140° are $\underbrace{12}_{\text{Number of pulses per motor rotation}} \times \underbrace{64}_{\text{gear ratio}} \times \underbrace{\frac{238}{360}}_{\text{Motor output shaft rotation rate}} = 507.7\text{pulse}$

Figure 1.4 is the encoder output waveform of the handle motor.In order to improve the encoder resolution, this bicycle's control program samples the B-phase rotation angle at the rising and falling edges of the A-phase encoder pulse, and obtains twice the normal count value.Using ± 507.7 counts for a handle angle of ± 70° , handle angle control with a minimum angle of 0.14° is possible.

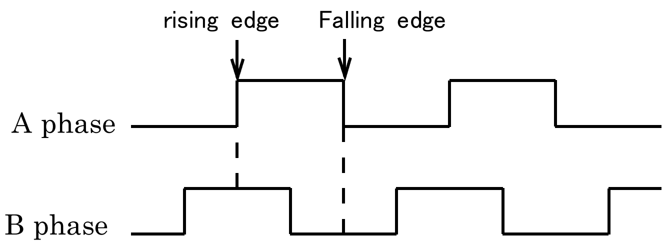


Figure 1.4 Encoder output waveform

◇ Let's try 1.12

Let's observe the output waveform of an incremental encoder to understand how it works.

Things to prepare: Automatic posture control bicycle, remote control, oscilloscope (one that can stop the image), 2 oscilloscope probes

Observation method: Observe the waveform between Pin 3 (A phase) and Pin 4 (B phase) of the handle motor connector CN4 and GND. The vertical axis of the oscilloscope measures 2V/div, the horizontal axis measures 2ms/div, and a normal single trigger. Turn on the power to the bicycle and remote control, and perform the work in the following order.

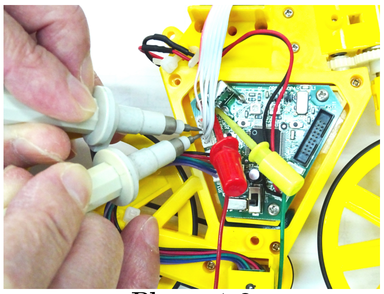
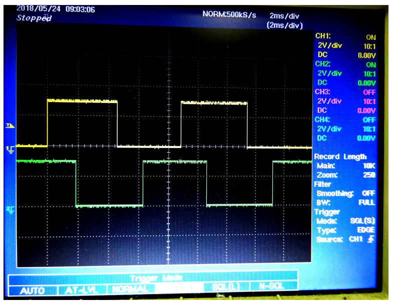


Photo 1.3



2018/05/24 09:03:06
Stopped
NORM:500kS/s 2ms/div (2ms/div)
CH1: ON 2V/div 10:1 DC 0.00V
CH2: ON 2V/div 10:1 DC 0.00V
CH3: OFF 2V/div 10:1 DC 0.00V
CH4: OFF 2V/div 10:1 DC 0.00V
Record Length Main: 10K Zoom: 250
Filter Smoothing: OFF BW: FULL
Trigger Mode: SGL(S) Type: EDGE Source: CH1

Photo 1.4

Observation method: Observe the waveform between Pin 3 (A phase) and Pin 4 (B phase) of the handle motor connector CN4 and GND. The vertical axis of the oscilloscope measures 2V/div, the horizontal axis measures 2ms/div, and a normal single trigger.

Turn on the power to the bicycle and remote control, and perform the work in the following order. ① Connect the ground leads of the two probes to the GND check pin on the board using pin clips. Photo 1.3 ② Remove the hooks and tips of the two probes and touch pins 3 and 4 of CN4 directly with the probe contacts. Photo 1.3 ③ Use the remote control to move the handle slightly to the left or right and observe the A-phase and B-phase waveforms at the same time. Photo 1.4

Note: Divide the work among multiple people: the person holding the probe, the person operating the remote control, and the person operating the oscilloscope.④ Operate the handle left and right to observe the waveforms of phase A and phase B.As shown in Figure 1.5, when steering to the left, phase A lags, when steering to the right, phase A advances, and as the steering speed increases, the frequency of the output wave increases.

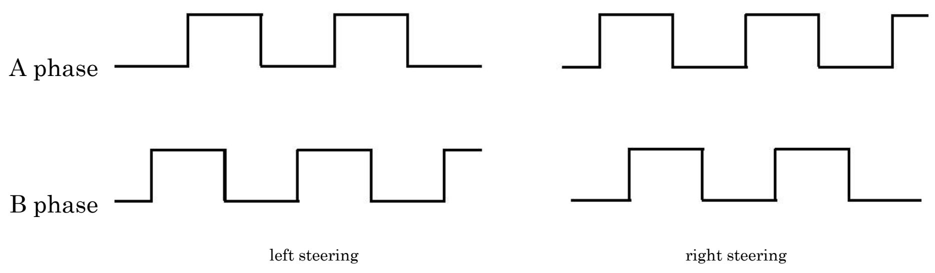


Figure 1.5 Encoder output phase waveform

1.3Pedal motor specifications and driving performance

Automatic posture control bicycle pedal motorDong Hui motor Industrial co.,Ltd(Built-in gear head and incremental encoder made in ChinaDCBrushed motor (photo)1.5)is. Show motor and encoder specifications1.3 Figure the connection diagram1.4It is shown in .

photograph1.5　Comes with brush for brushDCMotor

mold	GM12-N20VA-09220-150-EN	Reduction ratio	1/150
Rated voltage	6〔V〕	Rated current	120〔mA〕
Rated rotation speed	11700 [rpm] Motor rotation	Motor shaft torque	2.5 [g･cm]
No-load rotation speed	15000〔rpm〕	No load current	28 [mA]
Encoder pulse	3〔P/R〕	Output channel	90° phase A phase B phase
Encoder power supply	3.5 [V] ~ 20 [V]	Power current	5 [mA] ~ 10 [mA]

table1.3Pedal motor/encoder specifications

Other numbers regarding the running performance of bicycles are as follows. Rear wheel external size: 75mm

Motor output shaft pulley diameter: 22mm
Rear wheel side pulley diameter: 20mm
75 when the rear wheel rotates oncemm×π≒235mmThe vehicle moves forward. In order for the rear wheel to rotate once, the motor must

$\frac{20}{22}\times150136$　It rotates and the encoder pulse outputs 409 pulses.

When the bicycle travels 1m $\frac{409\text{pulse}}{235\text{mm}}\times1000\text{mm}=1741$ A pulse is detected.

Since the pedal motor is designed for forward movement only, the encoder uses only the A phase to calculate speed.
This vehicle speedVThe number of pulses proportional to the handle angle μ for automatic attitude control is calculated as shown in equation 1.4.

$$\mu = K\frac{\tan\theta}{V^2} \cdots \text{formula 1.4}$$

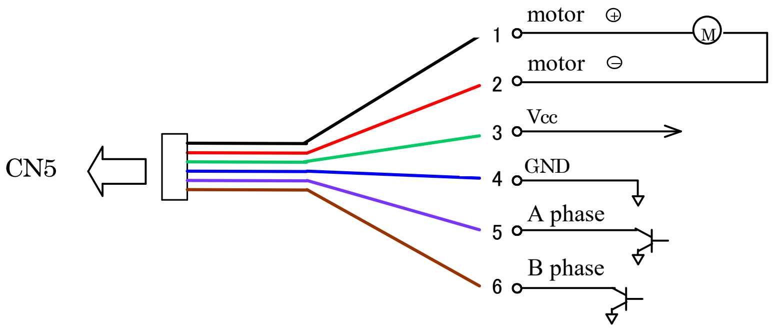


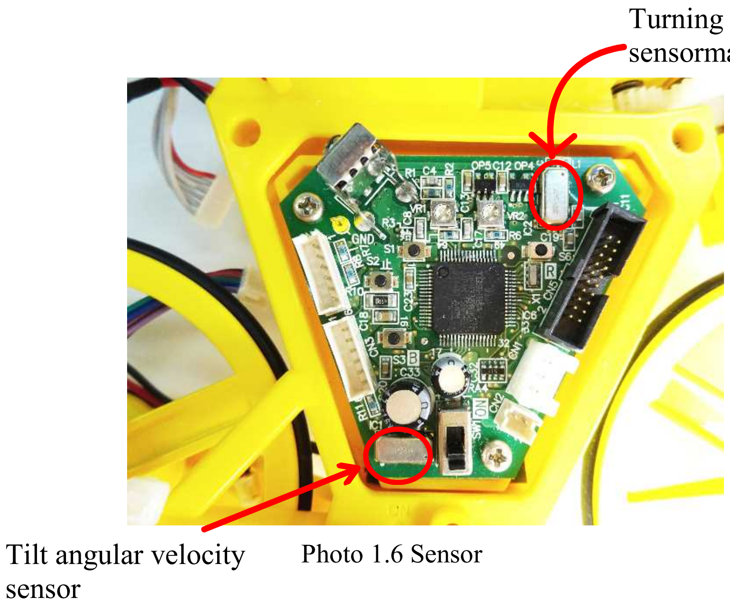
figure1.6 Pedal motor/encoder connection diagram

1.4Anti-vibration measures

Automatic posture control bicycle has photo sensors that detect tilting and turning1.6 It is installed on the control board as shown in the figure.
This sensor is easily affected by vibrations transmitted by wheels and motors, and vibrations become electrical noise, which causes large errors in attitude control calculations.

However, if you increase the amount of vibration isolating material to an extreme in order to eliminate vibrations, there will be a time delay in the sensor's response to actual tilting or turning, making it impossible to control the attitude.

For this bicycle, the entire control board with the sensor installed is photographed.1.7It is lifted from the car body using alpha gel vibration damping material with double-sided tape.



Turning angular velocity sensormaterial
Tilt angular velocity sensor

Photo 1.6 Sensor

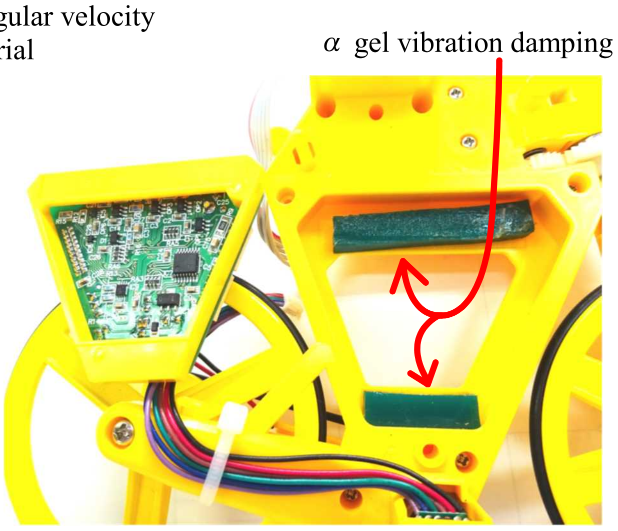


Photo 1.7 α gel vibration isolation material

Chapter 2 Board mounted devices/hardware

2.1angular velocity sensor

2.1.1Principles, usage, and problems of vibration type angular velocity sensors

The Murata Manufacturing Co., Ltd. angular velocity sensor (ENC-03 R□-R) used this time is a vibration type angular velocity sensor.

Vibrating bodies, pendulums (Foucault's pendulum), spinning tops (mechanical gyro), etc. try to maintain their current absolute angles. For example, as shown in Figure

Figure 2.1 Absolute angle
ofEarth Sesame

2.1, when a globe placed on the earth is viewed from space, the axis of the top remains in the same direction regardless of the rotation of the earth, but when viewed from the earth's coordinates, the top's axis changes by the angle of rotation. The axis is moving. This is a gyroscope.
In the case of a vibrating gyro, which operates on a principle similar to Foucault's pendulum, the piezoelectric vibrating body tries to maintain its current absolute angle, but when an external force with an angular velocity is applied to it, a Coriolis force proportional to the angular velocity acts. Masu. A vibration-type angular velocity sensor is a sensor that generates a voltage proportional to this Coriolis force.

However, the output signal of this angular velocity sensor is Drifts affected by temperature changes are large Also vibrates the sensor30kHzContains a lot of noise before and afterTherefore, some ingenuity is required in order to use the angular velocity signal output from the sensor.

Idea ① Low pass filter

Amplify the sensor signalOPAmplifier (schematic diagram)OP02･OP03)On the negative feedback side1000pFA capacitor applies feedback to suppress the amplification of high frequencies, creating a low-pass filter.

Idea ② Automatic drift correction

Tilt angular velocity sensor signalOPAmplifier (schematic diagram)OP01･OP02･OP03)DC analog amplification is performed by several hundred times.
In order to correct the drift of the angular velocity sensor in this DC amplification, the serial inputDAC (Digital to analog converter)IC2usingCPUThe analog input (P51/AN1)The center of the signal is within the analog input range (0V～3.3V)Near the center of (1.65V)so that it becomesCPUOutput the correction value from the program from the side.OP02ofFourAnalog addition is performed at pin No. 3 to cancel the drift.

Related pages:『2.2serialDAC”『5.1.Initialization of analog values”

2.1.2Output characteristics of vibration type angular velocity sensor

Vibration type angular velocity sensor (ENC-03□-R)Photograph the outline of2.1Shown below. This sensor is2To reduce interference between each sensor assuming that it will be used in an axis2Two oscillation frequencies are available.

mold given name	Oscillation frequency
ENC-03RC-R	30.8kHz
ENC-03RD-R	32.2kHz

Next, we will explain the rotation angle and analog output of this angular velocity sensor. Diagram on angular velocity sensor2.2Diagram when giving a rotation angle like2.3You will get a sensor output like this.

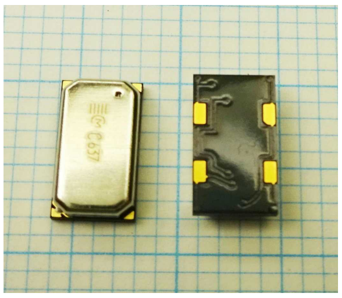


Photo 2.1 Angular velocity ensor

As shown in ① in Figure 2.3, perform constant angular velocity movement up to a rotation angle of 180° , then stop for a certain period of time, perform constant angular velocity movement in the opposite direction with a rotation angle of 360° , and after stopping for a certain period of time, return to the original position at a rotation angle of 180°. It returns with constant angular velocity motion.

At this time, the angular velocity is ② and the angular acceleration is ③, but the analog output of the ④ sensor has a waveform that is halfway between the angular velocity and angular acceleration.
This sensor's analog output saturates when constant angular velocity motion continues (1 to 2 seconds), so it is not suitable as a control sensor for large bicycles (such as 26-inch models) that have a slow inherent shaking cycle.

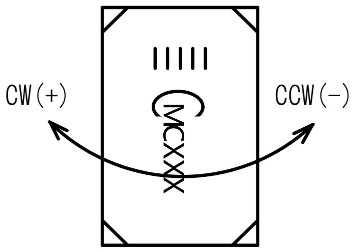


Figure 2.2 Sensor rotation direction

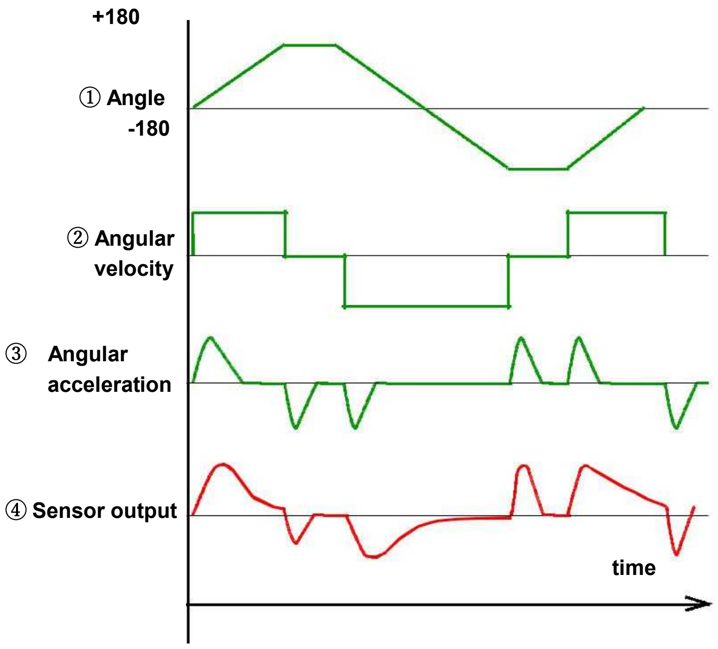


Figure 2.3 Sensor output waveform

◇ Let's try 2.1 Let's check the characteristics of the angular rate sensor in the order shown below. In the case of tilt sensors, the OP amplifier 3 output pin 1 is the easiest point to see.

(1) Turn on the bicycle and remote control and warm up for about 10 minutes.

(2) Press the stop button switch on the remote control to perform automatic drift compensation.

(3) After the flicker stop of the L1 lamp, apply the oscilloscope probe contact directly between Pin 1 and GND of OP amplifier 3 and observe the output wave while rocking the bicycle. At this time, care must be taken to ensure that the probe contact does not deviate from the first pin of the OP3. If the probe of the OP3 Pin 1 seems to be difficult to hit, a check pin can be installed. Now that the DIP components are gone, even a board modification such as adding a check pin requires a little technology.

OPamp 3 1st pin land shown in photo 2.2 to photo 2.3 at 1.5mm square ϕ Solder a check pin with a 1mm hole as shown in picture 2.4.

The hook tip of the probe can now be connected, but the land is not very strong, so be careful not to apply strong force and measure.

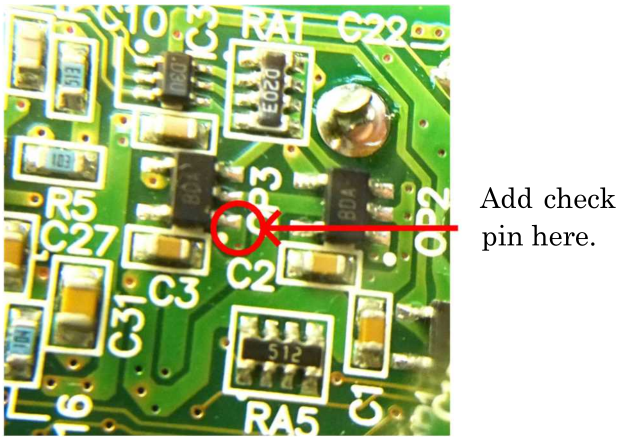


Photo 2.2 Check pin on OP3

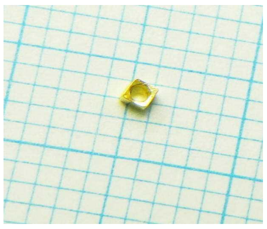


Photo 2.3 Check pin

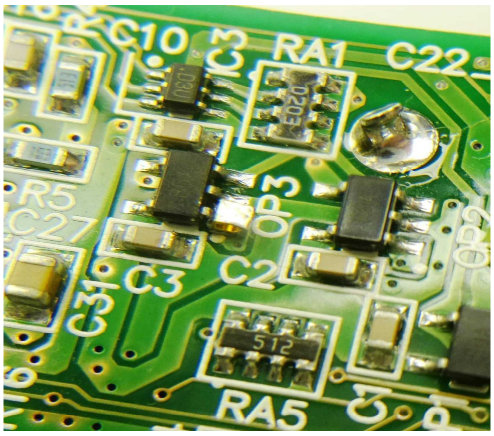
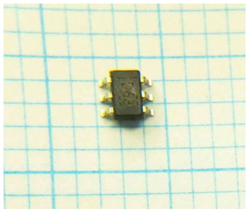


Photo 2.4 Check pin installation completed

2.2 Serial DAC IC2, IC4 (900 version) IC3, IC4 (ARM version)



The control board for this bicycle uses the Analog Devices serial interface DAC AD5611BKSZ (Photo 2.5). Its main specifications are shown in Table 2.1.

Photo2.5 Serial Interface DAC

Item	Contents
power supply	2.7V to 5.5V Current consumption 100μA
Conversion Accuracy	10bitDAC
DA output voltage	0V to VDDRL=2kΩ Rail-to-Rail
DA output voltage	0.5V / μs
DA conversion method	Resistor string method
Serial Clock	30MHzmax

Table 2.1 Specifications of AD5611BKDZ

This serial interface is connected to the CPU via synchronous serial communication, and outputs the automatic drift correction value of the angular rate sensor calculated in the CPU to the adder circuit of the sensor amplifier circuit, automatically correcting the drift of the angular rate sensor. See Section 2.3.3.

The timing diagram for serial synchronous communication is shown in Figure 2.4. When SYNC is LOW, data is active and data is read at the falling edge of the clock SCLK. Data is read from the MSB side, and after the LSB is read, it is output to the DAC register.

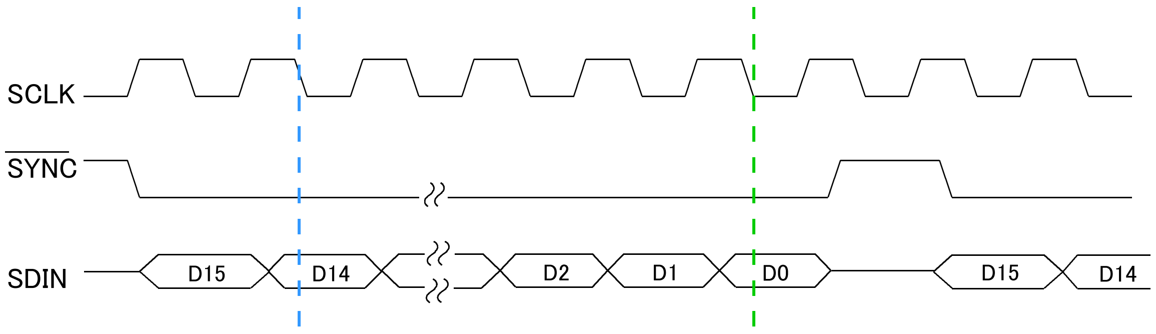


Figure 2.4. Synchronous communication timing diagram

The technical details of the resistor string DAC, R/2R ladder DAC and ADC will be explained later.

2.3 Analog（ op-amp ） circuits

2.3.1　Rail- to -Rail Op Amps

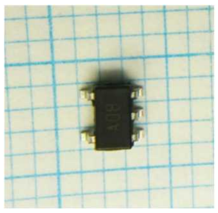


Photo 2.6 Operational amplifier AD8515

uses the Analog Devices op-amp AD8515 (Photo 2.6) . This op-amp is a very easy-to-use element that has rail -to- rail input and output and can operate on a single power supply. Here, we will first explain the input and output operating ranges, which are the basis of how to use an op-amp.

Input side operating range:

A typical input circuit of an operational amplifier is a differential amplifier circuit as shown in Figure 2.5 .

In this circuit, the input Vin requires a level that is VBE (dead band of approximately 0.6V) higher than VEE, and since there is at least a loss (dead band) on the Vcc side due to the current mirror circuit, the operating range of Vin is between (Vcc - dead band) and (VEE + dead band) .

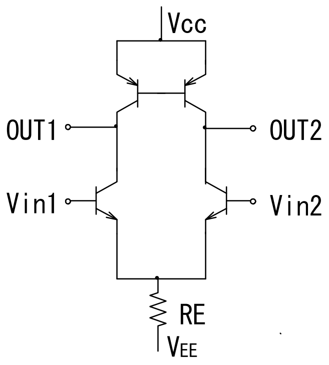


Figure 2.5 Operational amplifier input circuit

Output side operating range:

A typical op amp output circuit is a complementary circuit as shown in Figure 2.6 . In this circuit diagram, the output voltage swing range is (Vcc-V BE) to (V EE +V BE) .

As you can see, even if we look at only the input and output circuits, we cannot simply use the full power supply voltage.

DC amplifier with multiple stages of transistors directly connected , voltage losses occur in circuits other than the input and output circuits as well . As shown in Figure 2.7 , the operating range of a general-purpose op amp is (Vcc - 1.5V) to (V EE + 1.5V) .

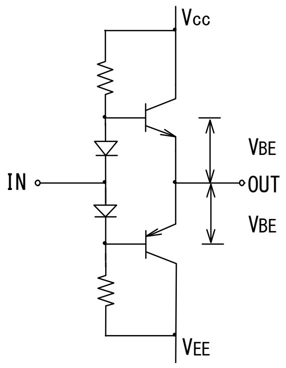


Figure 2.6 Operational amplifier output circuit

Op-amps that can be operated with a single power supply can be used up to the full power supply voltage on the lower side, but there is a non-usable range of about 1.5V on the Vcc side .

Rail -to- rail (full to full) devices use the full power supply voltage, but there are also elements that are rail -to- rail on the output side only and rail -to- rail on both the input and output sides , so please check the specifications on the data sheet when selecting elements.

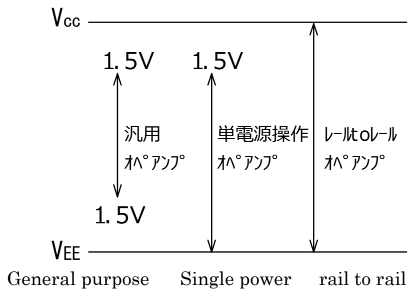


Figure 2.7 Operating range of operational amplifier

2.3.2 Operational amplifier circuit

This section provides a general explanation of the operational amplifier circuit used in the control board of an automatic attitude control bicycle.

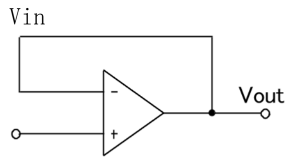
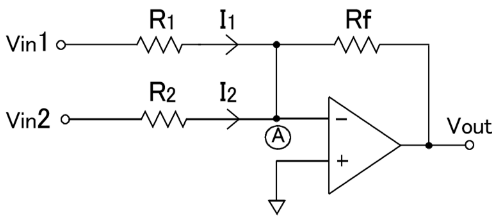


Figure 2.8 Follower circuit

・Follower circuit (Figure 2.8)

When using an operational amplifier with negative feedback, the input terminals (-) and (+) are virtually shorted and have the same potential. The follower circuit feeds back 100% of Vout to the (-) input terminal, so Vin = V(-) = Vout, and the amplification degree is 'I', which is non-inverting amplification.
Followers are used to convert high-impedance circuits that cannot absorb energy into low-impedance circuits that can conduct current.



Addition is performed based on A,

Based on A $\frac{R_f}{R_1}$ is inverted amplified.

Figure 2.9 Adder circuit

・Addition circuit (Figure 2.9)

Since point A of the output of the adder circuit is virtually grounded,

$$V_{out} = -R_f(I_1 + I_2) \quad \cdots\text{formula2.1}$$

$$= -R_f\left(\frac{V_{in1}}{R_1} + \frac{V_{in2}}{R_2}\right) \quad \cdots\text{formula2.2}$$

$R = R_2$ Under the conditions of

$$= \underbrace{-\frac{R_f}{R_1}}_{\text{Amplification}} \underbrace{(V_{in1} + V_{in2})}_{\text{addition}} \quad \cdots\text{formula2.3}$$

Addition in equation 2.3 is performed based on A point.
The virtual ground potential of ○A can be changed arbitrarily by changing the potential of the input terminal (+).

・inversion amplification

Figure 2.10 shows an inverting amplifier circuit with VR for variable amplification and capacitor C1 for high-cut filter. The amplification factor can be varied from 10k/10k=21 times to 200k/20k=10 times.
For filter characteristics, calculate the frequency where Rf=XC1.

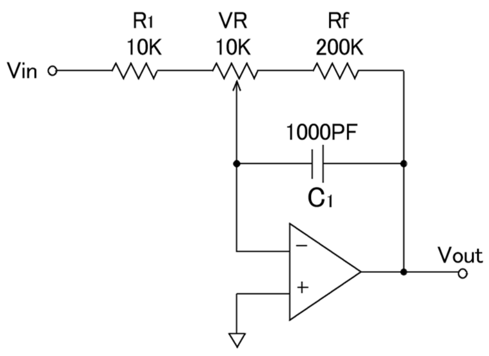


Figure 2.10 iterative amplification

$$\frac{1}{2\pi fc} = R_f$$ R ···Find f from equation 2.3.

$$f = \frac{}{2\pi R_f C} = 800\text{Hz} \cdots\text{formula 2.4}$$

The filter effect starts to appear around 800Hz.

2.3.3 Analog circuit configuration

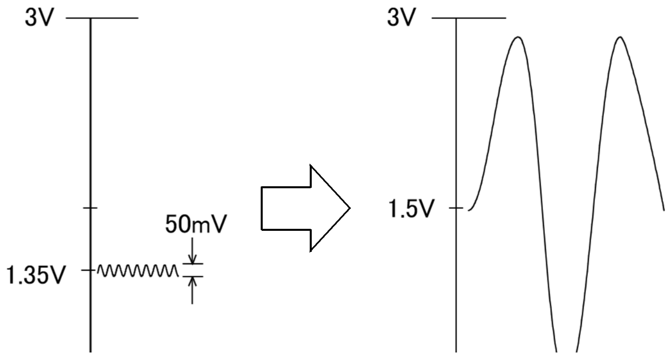


Figure 2.11 Analog signal size amplification

Analog amplify the angular velocity sensor signal from Section 2.1 using the operational amplifier described in Section 2.3 and input it to the ADC built into the CPU.
The magnitude and characteristics of the signals handled by this analog amplifier circuit are explained using Figure 2.11.

sensor output voltage

Reference output: 1.35V±0.15V
±0.15V is the output change due to temperature drift, etc.
Sensor signal: The maximum change level of the output signal when the bicycle is running is about 50mVP-P. The sensor signal contains a lot of 30kHz carrier noise and vehicle body vibration noise.

CPU analog input

The analog input range on the CPU side is 0 [V] to 3 [V], so the best situation is for the signal to swing by a maximum of ±1.5 [V] around 1.5 [V].
∴This sensor signal amplifier must be designed to comply with the following items.
・±0.15V temperature drift countermeasure
・DC amplifier with an amplification factor of about 100 times
・Noise countermeasures
The operation of this amplifier will be explained using the analog circuit diagram for tilt sensor shown in Figure 2.12.

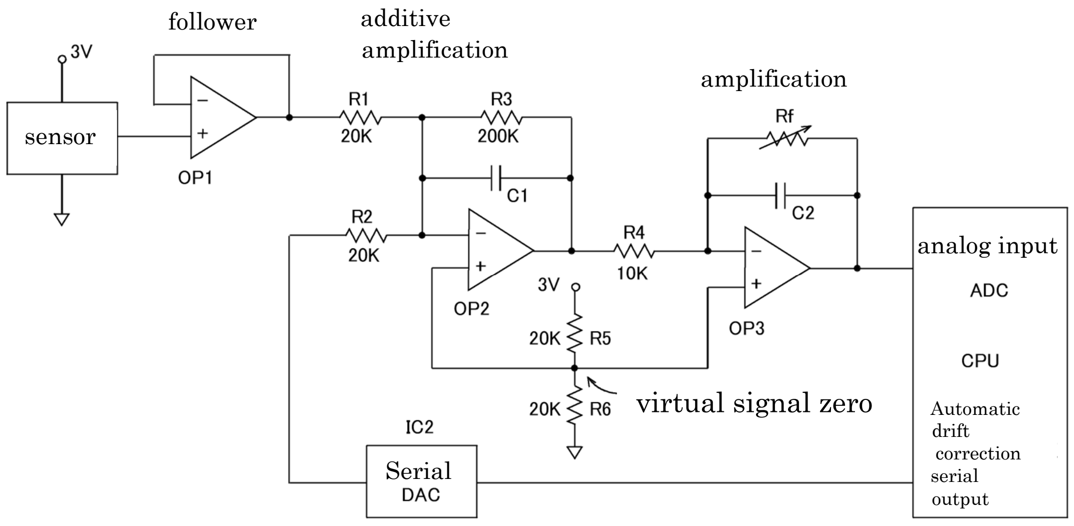


Figure 2.12 Analog circuit schematic diagram

・Follower OP1
Since the output impedance of the sensor is high, a follower amplification is installed to lower the impedance. If the input resistor R1 of the summing amplifier is connected with high impedance, an error will occur in the summing operation.

・Additional amplification OP2
The sensor signal passed through the follower and the automatic drift correction value output from the CPU are added here, and the center of the angular velocity signal is adjusted to a potential near the center of the amplification range.

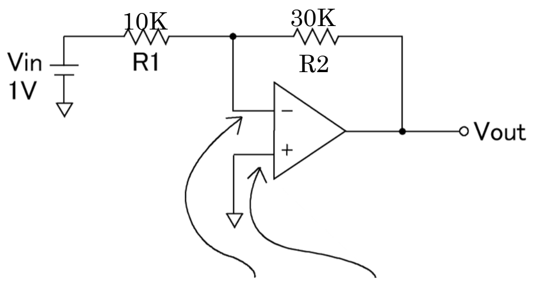
・Serial interface DAC IC2
A DAC (AD5611BKSZ) connected to the CPU via synchronous serial communication converts the automatic drift correction value into digital to analog and passes it to the summing amplifier.

・Virtual signal zero
Using R5 and R6, connect 1.5[V], which is obtained by dividing the 3[V] power supply, to the (+) terminals of OP2 and OP3, and set 1.5[V] as the signal zero of this amplifier circuit, 1.5[V] ±1.5[V] is the amplification range.

・Amplification OP2・OP3
OP2 is a 10x fixed gain and OP3 is a variable gain amplifier circuit.
Capacitors C1 and C2 are attached to the negative feedback side to form a high-cut (low-pass) filter.

・Automatic drift correction
A correction value is calculated in the CPU so that the average value of the analog input to the CPU is around 1.5 [V], and is output to the OP2 adder circuit via the serial DAC.

◇Try it 2.2
Let's understand the basic operation of an operational amplifier.
We will explain operational amplifier feedback using the most basic inverting amplifier circuit shown in Figure 2.13



When the circuit is balanced, the potentials of the (-) and (+) terminals are equal. This is called virtual ground or virtual short circuit.

Figure 2.13 Inverting amplifier circuit

An operational amplifier amplifies the voltage difference between the input (-) and (+) terminals by more than 100,000 times, so if you apply feedback from Vout to the (-) terminal with R2 as shown in Figure 2.13, the (-) terminal and (The circuit is balanced when the potential difference between the +) terminals disappears.
When the circuit is balanced, the potentials of the (-) and (+) terminals are equal. This is called virtual ground or virtual short circuit.

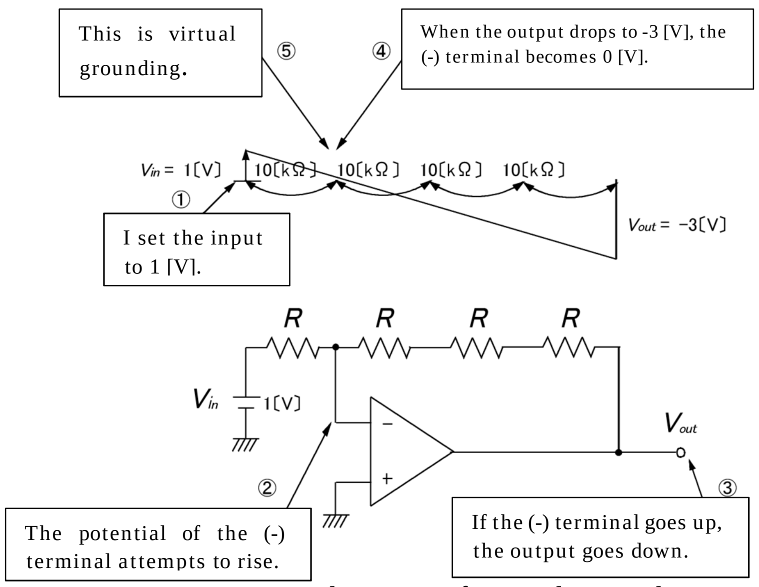
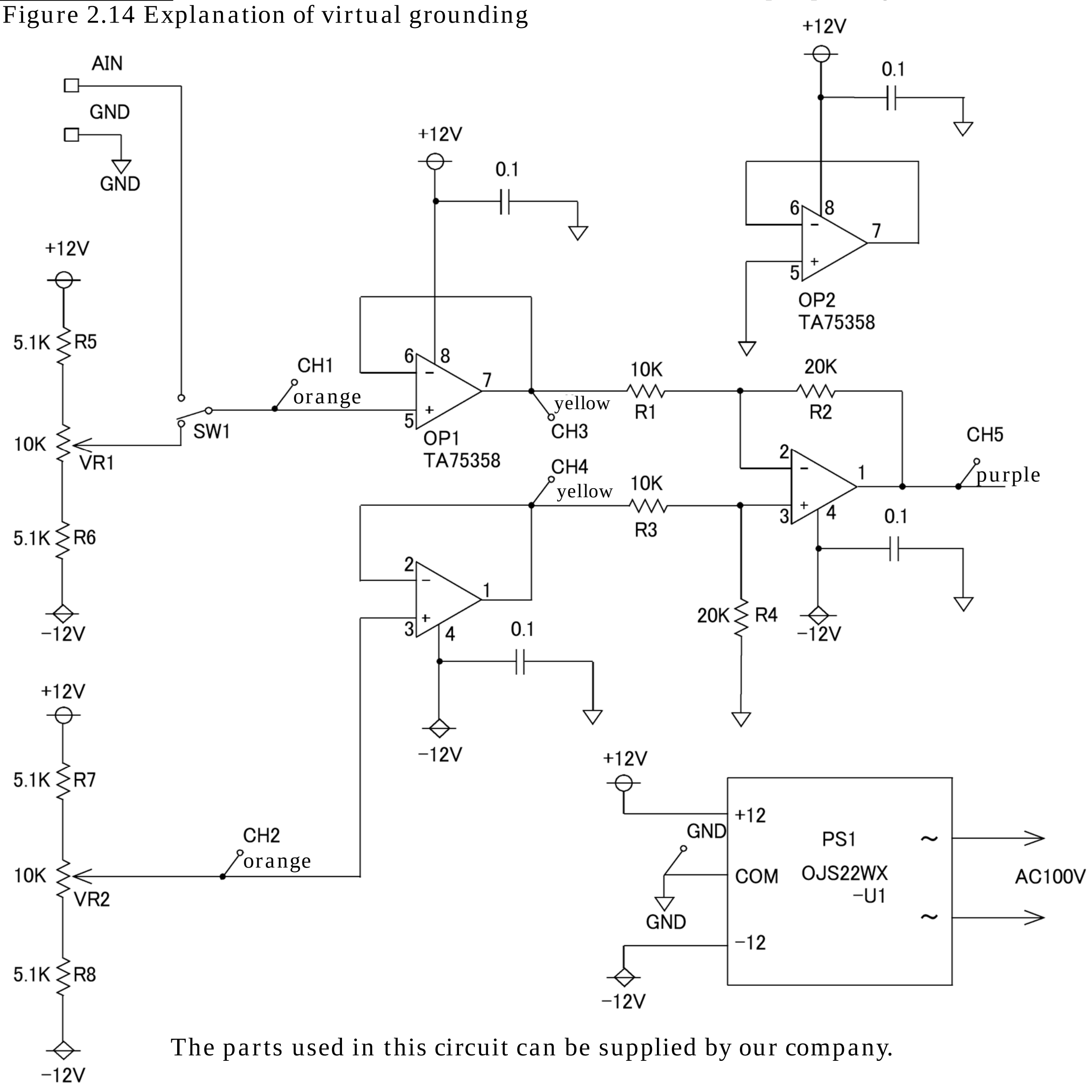

This is virtual grounding.
⑤
④
When the output drops to -3 [V], the (-) terminal becomes 0 [V].
V_{in} = 1[V]
10[kΩ] 10[kΩ] 10[kΩ] 10[kΩ]
①
I set the input to 1 [V].
V_{out} = −3[V]
R R R R
V_{in}
1[V]
V_{out}
②
③
The potential of the (-) terminal attempts to rise.
If the (-) terminal goes up, the output goes down.

Figure 2.14 Explanation of virtual grounding

If Figure 2.13 is represented as a seesaw diagram of the resistance ratio of R1 and R2, it becomes as shown in Figure 2.14. Since the (+) terminal is grounded and has 0V, the (-) terminal must also be grounded at 0V. It works in order.

In order to confirm the basic operation of an operational amplifier, let's create the operational amplifier test circuit shown in Figure 2.15 on a bullet board and measure the input/output characteristics and frequency characteristics.

The operational amplifier used in Figure 2.15 is a general-purpose product that has two operational amplifier circuits built into an 8-pin package.



The parts used in this circuit can be supplied by our company.

The circuit configuration is such that the first stage (OP1) converts the outputs of VR1 and VR2 into impedances using a follower, so that the next stage's operational amplification calculation can be performed without error. The second stage is a differential amplification with a gain of 2x.
Please follow the steps below to measure.
Measurement ① Inversion amplification
⊖Vary the input voltage and measure the change in the output voltage.
Turn VR2 to set check terminal CH2 to 0 [V].
Set SW1 to the lower side and turn VR1 to change CH1 from -6 [V] to +6 [V], measure the voltage of output CH5 at that time, and plot it on a graph. Since it is an inversion amplification, the result shown in Figure 2.16 can be obtained.

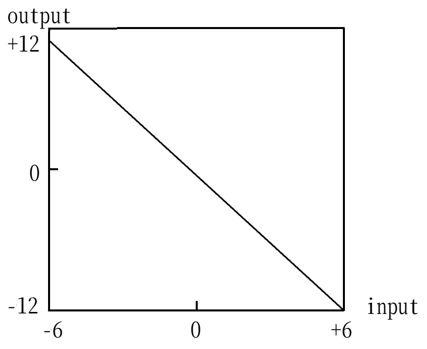


Figure 2.16 Inversion amplification characteristics

Measurement ② Non-inverting amplification
⊕ Vary the input side voltage and measure the change in the output voltage.
Turn VR1 to set check terminal CH1 to 0 [V].
Turn VR2 to vary CH2 from -6 [V] to ±6 [V], measure the voltage of output CH5 at that time, and plot it on a graph.

Measurement ③ Inversion amplification with offset
⊕Give an offset of +1 [V] to the input side, ⊖ Vary the input side voltage, and measure the change in the output voltage.
Turn VR2 to set check terminal CH2 to +1 [V].
With SW1 at the bottom, turn VR1 to change CH1 from -6 [V] to +6 [V], measure the voltage of output CH5 at that time, and plot it on a graph.

Measurement ④ Non-inverting amplification with offset
⊖Apply an offset of +1 [V] to the input side, ⊕ Vary the input side voltage, and measure the change in the output voltage.
Turn VR1 to set check terminal CH1 to +1 [V].
Turn VR2 to vary CH2 from -6 [V] to ±6 [V], measure the voltage of output CH5 at that time, and plot it on a graph.

Measurement . Frequency characteristic confirmation
Check the frequency characteristics that can be amplified by the operational amplifier.

The operational amplifier used in this circuit is of a type that cannot amplify very high frequencies.

Turn VR2 to set check terminal CH2 to 0 [V].
Connect a function generator between AIN and GND with SW1 on the top side, and give an AC signal of about P-P 10 [V].
Connect an oscilloscope to output terminal CH5 and observe the waveform.
Gradually increase the frequency without changing the input AC voltage and observe the change in the output waveform.

If the operational amplifier you are using is TA75358, waveform distortion will start around 20kHz and the limits of wide frequency range will become visible.

Column 2.1 Brushed DC Motor

Brushed DC motors are small, inexpensive, have a large starting torque, and are very easy to use. Using the schematic diagram of a brushed DC motor shown in Figure 2-A, we will explain how the motor rotates.

・How the motor rotates

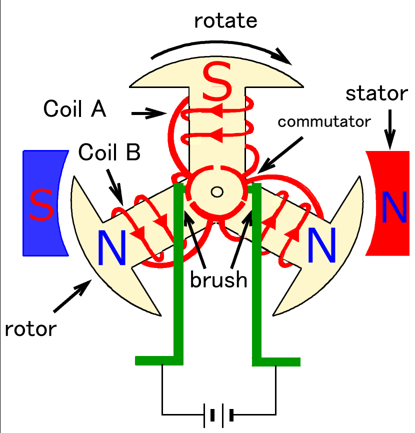


Figure 2-A brushed DC motor

The coil and commutator wound around the rotor shown in the figure rotate.
The permanent magnets on both sides of the diagram are fixed to the motor case and are called the stator. The instantaneous positions of the brush and commutator shown in the figure are such that current flows through coil A in the direction of the arrow, magnetizes the rotor as shown, and interacts with the stator's permanent magnet to rotate coil A to the right.
When it rotates approximately 90 degrees, the commutator's position changes, and current flows through coil B, causing it to rotate to the right in the same way. The rotor continues to rotate by repeating this action.

・Motor speed and torque

When the rotor rotates, the rotor coil moves in the magnetic field created by the stator's permanent magnets, creating an electromotive force in the rotor coil.Although it is a motor, it is also a generator, and as the rotation increases, the generated voltage also increases.

In this case, the generated voltage E_M and motor current I_M are shown below.

$E_M = K \cdot R \cdot B$ 〔V〕 ··· Equation 1 A voltage proportional to the rotation speed of the rotor is induced.

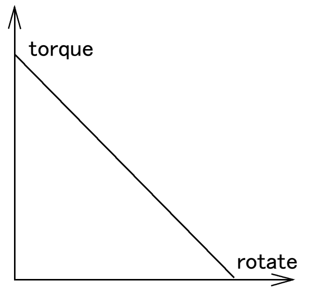
magnetic flux density
Rotor rotation speed
Other constants

battery voltage

Motor power generation voltage

$$I_M = \frac{V - E_M}{r} \quad \cdots\text{formula 2}$$

rotor coil resistance



When the rotation of the motor increases and the voltage generated by the motor becomes equal to the battery voltage, there is a limit to the increase in rotation speed. Since the motor torque is proportional to the current, according to Equation 2 above, the maximum torque is when the motor rotation is zero, that is, at startup, as shown in the torque vs. rotation speed characteristic in Figure 2-B.

Figure 2–B Torque rotation speed characteristics of brushed DC motor

DC motors have a large starting torque and are inexpensive, so they are often used in moving parts of home appliances.

In addition, in hand drills and other applications where starting torque is important, we deliberately rectify alternating current into direct current and use brushed.

This is because AC induction motors lack starting torque.
Recently, trains have been equipped with inverters, but most trains from a while ago were powered by DC motors.

Streetcars still use DC motors. The reason is that the starting torque is large.

Column 2.2 Various motors

Figure 2.C classifies motors by type, rotation principle, and application.
Let's briefly explain the uses and characteristics of each motor.

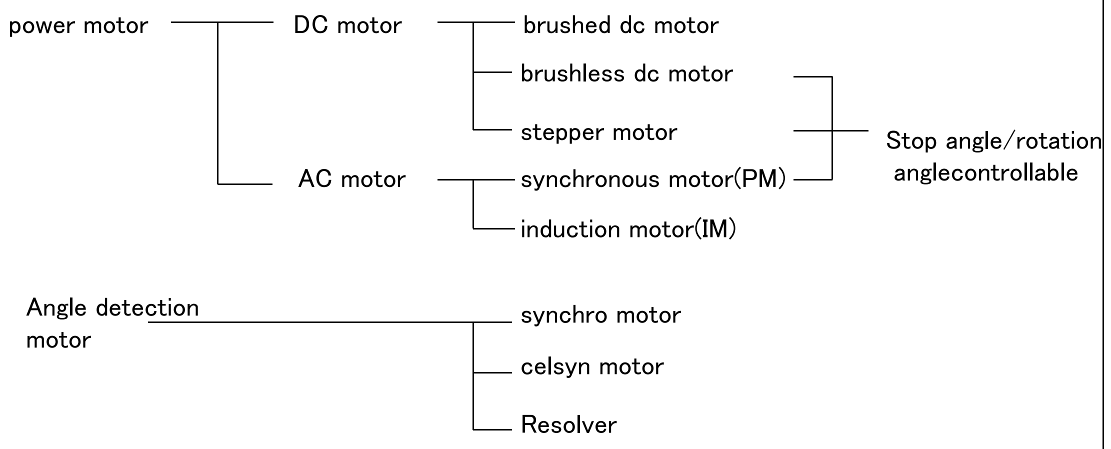


Figure 2. Classification of C motors

① Brushless DC Motor/Synchronous Motor (PM)
The brushless DC motor that rotates inside the hard disk and the synchronous motor that moves hybrid bicycles and large machinery in factories are actually motors with the same mechanism.

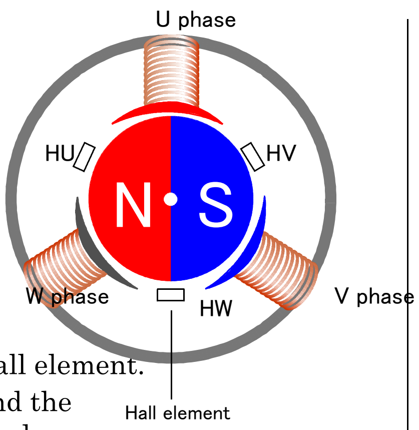


Figure 2.D brushless DC motor

As shown in Figure 2.D, the surrounding area (stator) consists of three-phase coils, and the central rotor consists of a permanent magnet (PM), and the current position (current angle) of the rotor is detected by a Hall element.
Since you know the position of the surrounding coils and the rotor position, you can know which coil to excite next and in which direction it will turn. It is also possible to stop at the target position if necessary. The rotation of the brushless DC motor/synchronous motor (PM) is completely synchronized with the command. The motor is driven by a dedicated motor driver or inverter. It will not turn even if you connect an AC power source or battery directly.

② Stepping motor

Since the rotation speed and rotation angle of a stepping motor can be directly controlled with a resolution of the number of steps, open-loop rotation control is easy.

However, since the rotation is a step motion, there are some aspects that are difficult to handle, such as vibration and resonance, and loss of synchronization where electrical and mechanical motions do not match.

Figure 2.E is a schematic diagram of a two-phase stepper motor.
The rotor is a magnet with fine teeth all around it.
The stator has A-phase and B-phase coils arranged in pairs around the entire circumference with 1/2 tooth mounting positions shifted.
Figure 2.E shows the state immediately after the B phase is driven to the suction side. Next, when the A phase is driven to the suction side, the rotor rotates to the right and advances one step. A special driver is required to rotate the stepping motor.

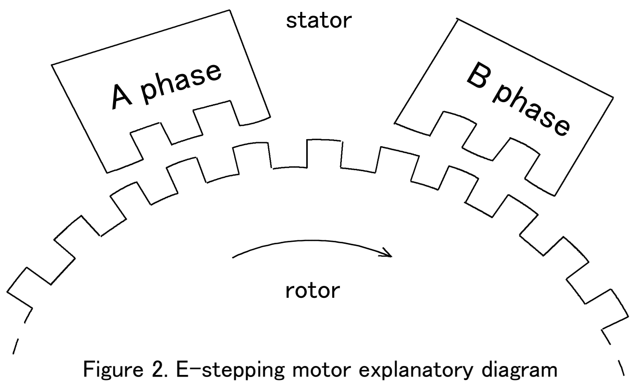


Figure 2. E-stepping motor explanatory diagram

③Synchro motor

Rather than a motor that converts electrical energy into mechanical energy, there is a motor that detects the angle and sends the rotation angle. Many people may not be familiar with them, but here we will introduce the most commonly used synchro motors.

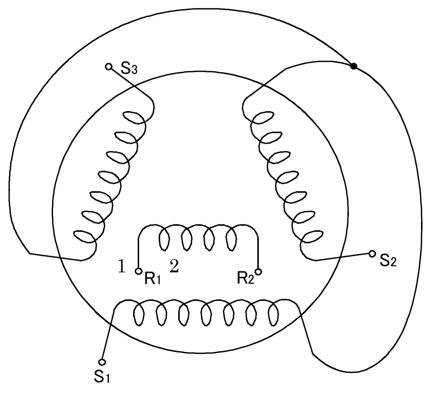


Figure 2F synchro motor structure

Figure 2F shows the structure of the synchronized motor.
The stator has three sets of coils located at 120 degrees each, which are star-connected and taken out to the outside. (S,S,S)123 A set of coils comes out from the rotor via a slip ring. Usually the rotor
It is used by adding AC power between R1 and R2.

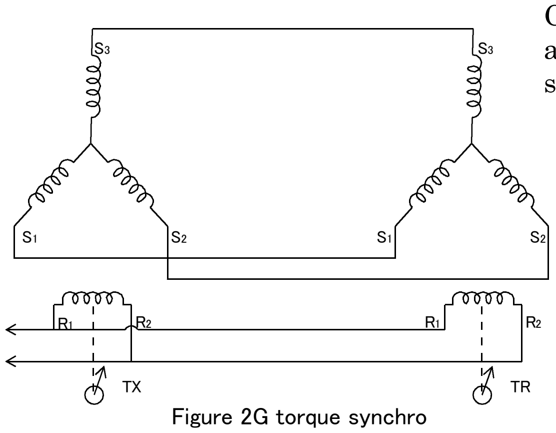
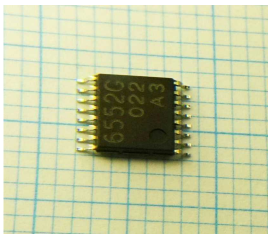


Figure 2G torque synchro

Connect the wires as shown in Figure 2G, and when you rotate the TX side, the TR side will also rotate by the same angle.

2.4 PWM control/H bridge driver

2.4.1 PWM control



The handlebars and pedals of the automatic attitude control bicycle are powered by a brushed DC motor with a built-in gearhead. This motor is driven by Toshiba's H-bridge driver IC (TB6552FNG) and is PWM controlled. The TB6552FNG contains two sets of circuits, A block and B block, as shown in the pin arrangement shown in Figure 2.17. Figure 2.18 illustrates the operation of a block on one side of the block using the block diagram for one circuit in the TB6552FNG and the terminal descriptions in Table 2.2.

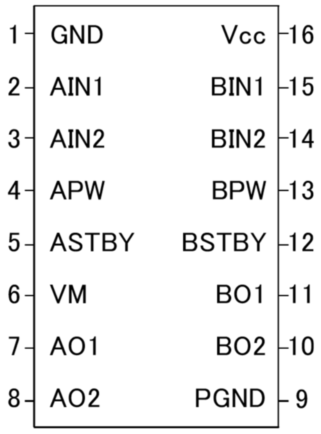


Photo 2.17 TB6552FNG terminal arrangement

IN1	Function selection of forward rotation, reverse rotation, short brake, and high impedance by combining N1 and IN2
IN2	
STBY	Active or standby switching of outputs
PWM	PWM modulation input terminal, creates a PWM waveform on the CPU side.
O1	Connect the output terminal motor
O2	Output is H-bridge operation or high impedance
VCC	Control power supply 2.7V to 5.5V
GND	Control GND
VM	motor power
PGND	Motor GND

Table 2.2 Terminal description

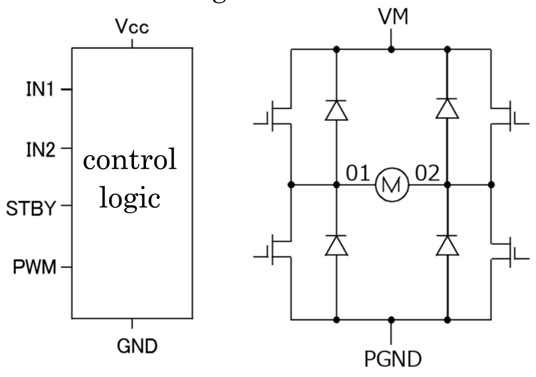


Figure 2. Block diagram of 18TB6552FNG

The TB6552FNG consists of an H-bridge and a control logic that drives the motor, as shown in Figure 2.18. The control logic includes function switching and PWM modulation input terminals, which are circuits that can control the rotation direction and rotation speed of the motor.

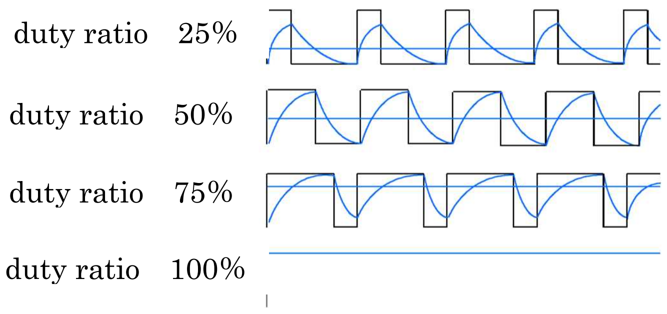


Figure 2.19 PWM

PWM（PluseWidthModulation） The control inputs a pulse with a variable duty cycle to the PWM pin, and controls the speed of the motor by changing the average value of the current flowing through the motor as shown in Figure 2.19. In Figure 2.19, the black line represents the input pulse waveform at the PWM terminal, and the blue wire represents the motor current.

2.4.2 H Bridge Operation

The operation of the H-bridge is then described using Table 2.3 and Figure 2.20. The red FET in Figure 2.20 represents the "ON" state, the red arrow represents the drive current of the motor, and the blue arrow represents the current due to the back EMF. When the motor is driven, the H bridge is controlled by PMW and repeats forward rotation A or reverse B and short brake C. However, if the system is instantly switched from the forward rotation A or reverse B state to the short brake C, a short circuit current will flow between the FETs due to the delay in the FET switching speed, which will cause heat generation in the FET. In order to prevent this instantaneous short circuit, a D or E state of about 300 ns is inserted during the switch from A or B to C. When switching from short brake C to A or B, the D or E state of about 300 ns is inserted in the same way as above. At this time, as indicated by the blue arrows of D and E, the current due to the back electromotive force generated in the coil of the motor flows to the flywheel diode. Operation diagram F is in a standby state.

control input				output			
IN1	IN2	STBY	PWM	O1	O2	action mode	Operation diagram
H	H	H	H	L	L	short brake	C
			L				
L	H	H	H	L	H	Reverse/forward	A
			L	L	L	short brake	C
H	L	H	H	H	L	Forward/reverse	B
			L	L	L	short brake	C
L	L	H	H	OFF		Stop	D E
			L	high impedance			
H/L	H/L	L	H	OFF		standby	F
			L	high impedance			

Table 2.3 Input/output functions

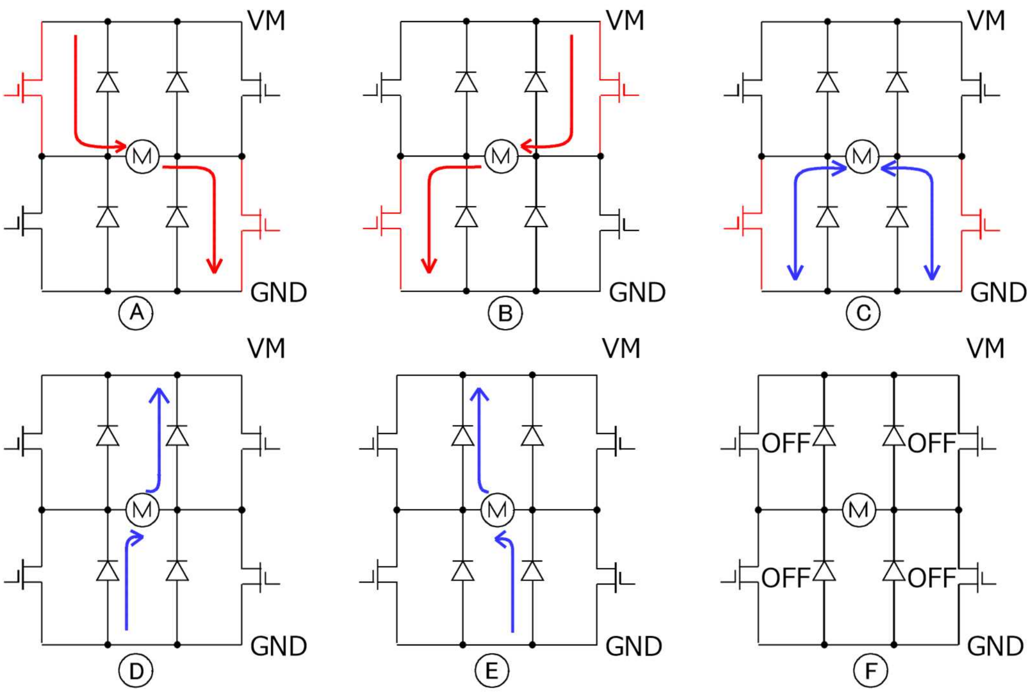
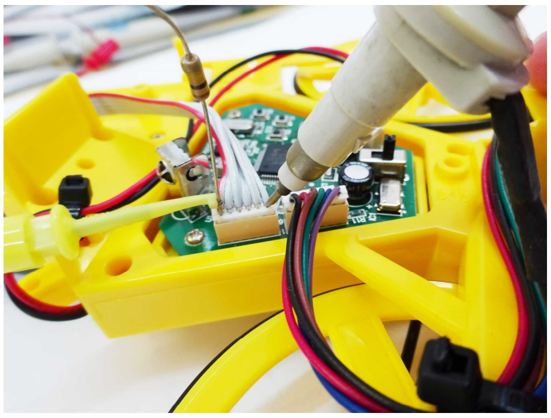


Figure 2.20 H-bridge operation explanation

◇Let's try it



2.3 Let's check the PWM control by observing the applied voltage waveform of the handle motor. The applied voltage waveform of the handle motor is the voltage waveform between O1 and O2 in Figure 2.18 and between O1 and O2 in Table 2.3. In the circuit, pins 1 and 6 of connector CN4 (Photo 2.8) are easy positions to apply the oscilloscope probe.

Photo 2.8 CN4 and probe

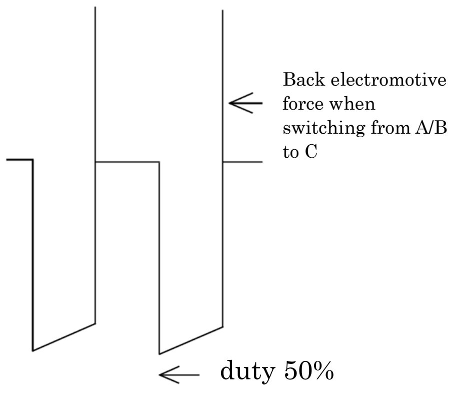
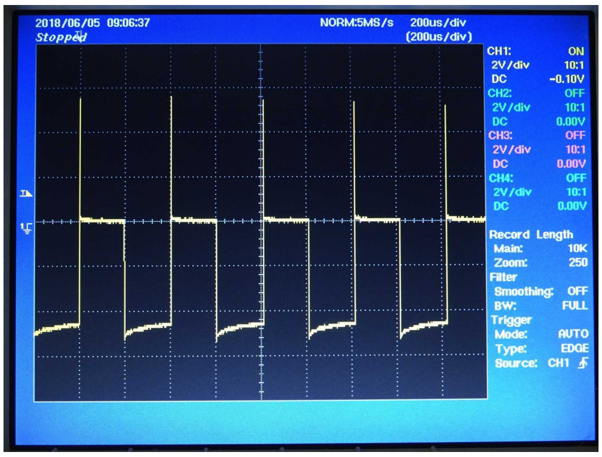


Figure 2.21 PWM control waveform

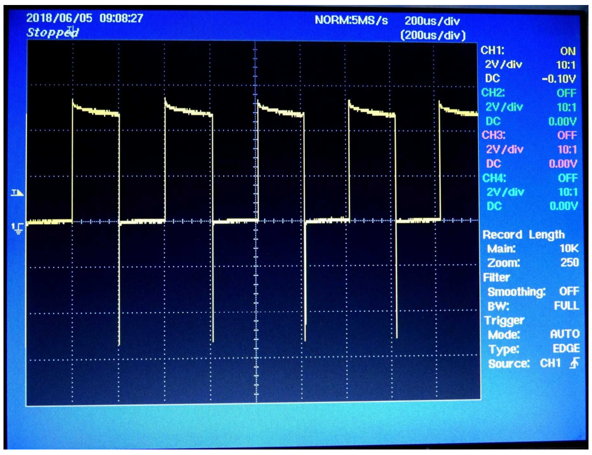
In Photo 2.8, the lead of the lead resistor is inserted into pin 1 of CN4 and pinched with a pin clip, and pin 6 is directly applied by the probe contact. The oscilloscope settings are vertical axis = 2 V/div, horizontal axis = 200 μs/div, auto mode. With the remote control and the bicycle turned on, the remote control is used to operate the steering wheel with the remote control, and the remote control is used to apply a deviation, and the voltage waveform is observed to change the duty until the handle follows.

Photos 2.9 and 2.10 show the waveforms as described in Section 2.4.2 above. The direction of the voltage is reversed on the right and left handles. If the deviation is large, the duty cycle will also be high, and if the handle follows until the deviation is zero, the voltage waveform will also disappear.



```
2018/06/05  09:06:37                NORM:5MS/s    200us/div
Stopped                                          (200us/div)
CH1:          ON
 2V/div     10:1
 DC       -0.10V
CH2:         OFF
 2V/div     10:1
 DC        0.00V
CH3:         OFF
 2V/div     10:1
 DC        0.00V
CH4:         OFF
 2V/div     10:1
 DC        0.00V
Record Length
 Main:       10K
 Zoom:       250
Filter
 Smoothing:  OFF
 BW:        FULL
Trigger
 Mode:      AUTO
 Type:      EDGE
 Source:     CH1
```

Photo 2.9 Right-hand drive operation



```
2018/06/05  09:08:27                NORM:5MS/s    200us/div
Stopped                                          (200us/div)
CH1:          ON
 2V/div     10:1
 DC       -0.10V
CH2:         OFF
 2V/div     10:1
 DC        0.00V
CH3:         OFF
 2V/div     10:1
 DC        0.00V
CH4:         OFF
 2V/div     10:1
 DC        0.00V
Record Length
 Main:       10K
 Zoom:       250
Filter
 Smoothing:  OFF
 BW:        FULL
Trigger
 Mode:      AUTO
 Type:      EDGE
 Source:     CH1
```

Photo 2.10 Left handle operation

2.6Remote control light receiving module

2.6.1Overview of light receiving module

Automatic attitude control bicycles are controlled remotely using an infrared remote control.

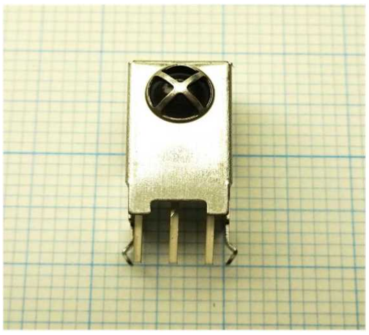


Photo 2.11 Light receiving module

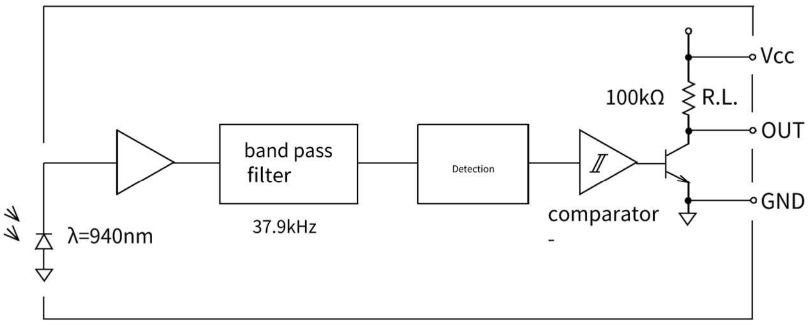
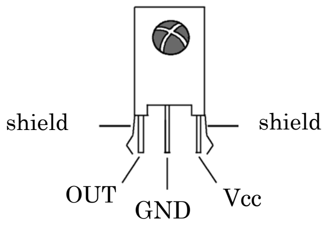


Figure 2.22 Light receiving module block diagram



Operating voltage	2.7V~3.6V
Consumption current	300µA
carrier frequency	37.9kHz
signal pulse width	400µs～800µs
Received light wavelength	940nm
usage environment	indoor
directional	H=45°V=35°

On the bicycle side is an infrared remote control receiver module from ROHM Co., Ltd.RPM7238-H5KType (photo)2.11)using.
This light receiving module is shown in the photo.2.11Shield case as well as figure2.22As shown in the block diagram of 37.9kHzEquipped with noise countermeasures such as a bandpass filter. Since the output is an open collector, multiple modules can be connected in parallel. For example, by attaching light receiving modules to the front and rear of a bicycle and connecting them in parallel, you can control the bicycle from all directions without being affected by the directivity of the remote control.
The signal waveform received by this module is shown in the figure.2.24as shown in37.9kHza career in 1200bps This is a waveform modulated with a serial signal. Signal when carrier is present1', if there is no carrier, the signal will be '0'.

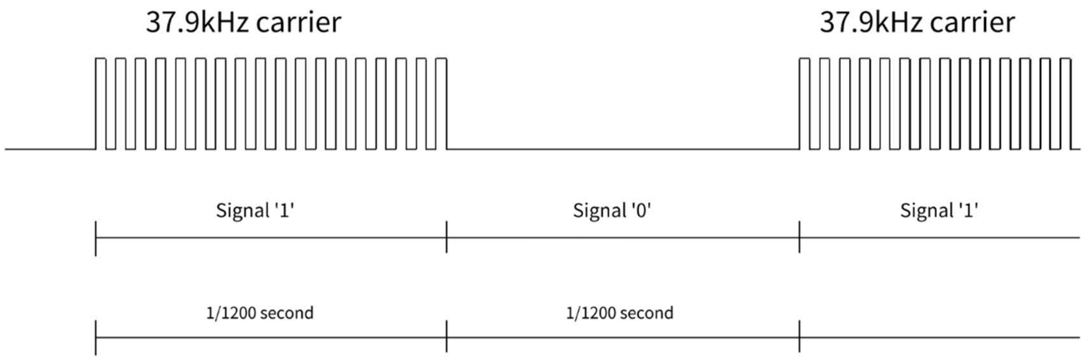


Figure 2.24 Signal waveform

2.6.2.How to use the remote control receiver module

figure2.25shows how to use the remote control receiver module. normal UARTAn infrared modem is added during communication.
on the sending sideUARTofTXDsignal and carrier37.9kHzofANDIt is modulated by an element and drives an infrared light emitting diode.
On the receiving side that receives infrared light,37.9kHzAfter removing noise with a band-pass filter and detecting it, the serial signal isUARTofRXDinput to the terminal.

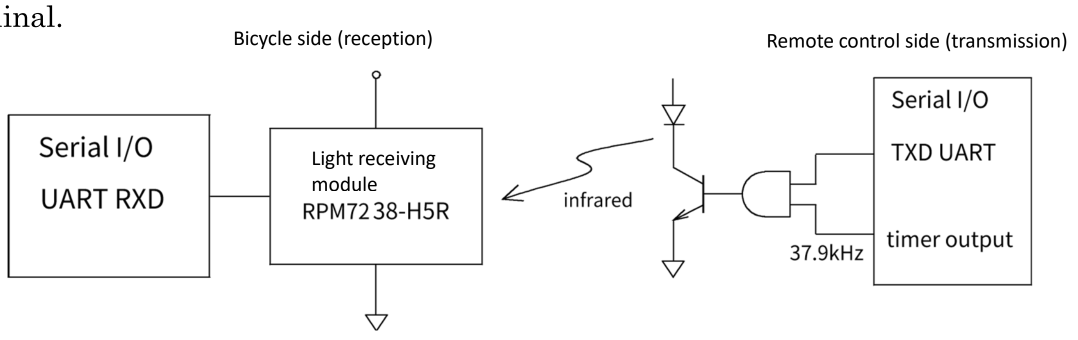


Figure 2.25 How to use the light receiving module

Optical communication is a communication method that is easily affected by noise, so please take measures such as adding a checksum to the message and other measures to prevent malfunctions from receiving noise from inverter-type fluorescent lamps.
This light-receiving module is of a type that can receive continuous serial signals, but the lightreceiving module of a general optical remote control is1There are many methods that can send only word-sized signals, so be careful when selecting a module.

2.6.3remote control transmitter

Diagrams information such as steering wheel angle, accelerator position, push button information, etc. from the remote control to the bicycle.2.26It is sent in this format.

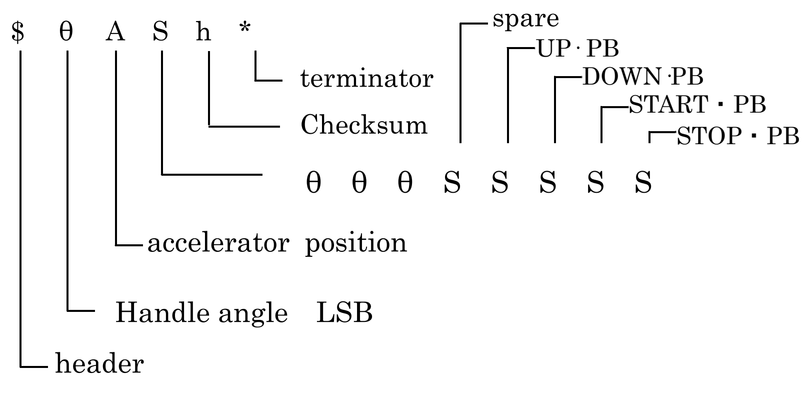


Figure 2.26 Transfer format

Transfer format is from header $ to terminator *6Fixed length in bytes. The checksum is θ ·A ·SThis is an exclusive or. Serial communication isUARTand the transfer baud rate is1200bpsis.

◇ Let's try it2.4

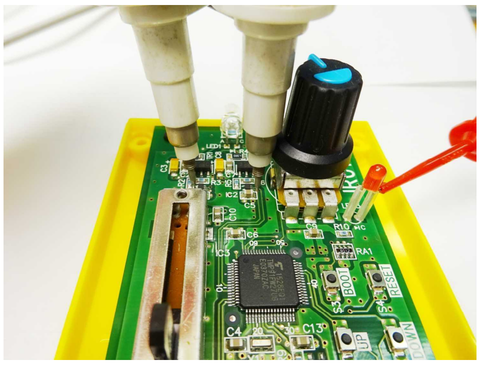
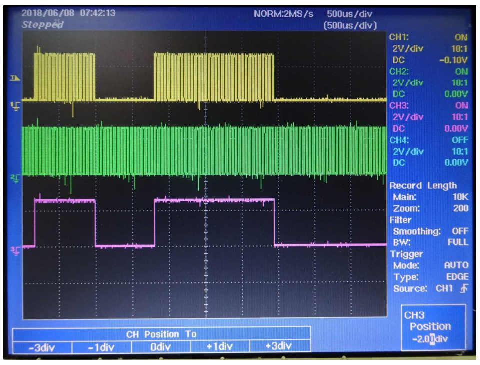


Photo 2.12 Remote control circuit check

The above diagram2.24Let's observe the modulation waveform of the remote control signal explained in . the remote control set screw6Remove the book, open the lid, and take a photo.

2.12Take a photo with the oscilloscope probe as shown.2.13Observe the modulation waveform. On the remote controlGNDThere is no check pin. photograph2.12 Like power pilotLEDPinch the outer lead of the pin clip.



```
2018/06/08  07:42:13          NORM:2MS/s   500us/div
Stopped                                    (500us/div)
CH1:          ON
 2V/div     10:1
 DC       -0.10V
CH2:          ON
 2V/div     10:1
 DC        0.00V
CH3:          ON
 2V/div     10:1
 DC        0.00V
CH4:         OFF
 2V/div     10:1
 DC        0.00V
Record  Length
 Main:       10K
 Zoom:       200
Filter
 Smoothing:  OFF
 BW:        FULL
Trigger
 Mode:      AUTO
 Type:      EDGE
 Source:     CH1
CH3
 Position
 -2.00div
         CH  Position  To
-3div  -1div  0div  +1div  +3div
```

Photo 2.13 Modulation waveform

The measurement points are on the circuit diagram.IC1orIC2 of1No. pin=UARTofTXD(waveform photo pink) and2No. pin =37.9kHzcarrier (waveform photo green) andFourNo. Pin = Modulation output (yellow waveform photo).
The vertical axis of the oscilloscope is 2V/div ·Horizontal axis 500μs/divMeasure with normal single trigger. photograph2.13 The waveform of the signal '1' '0' '1' '1' '0' '0'is.

◇ Let's try it2.5
Infrared remote controls are not very resistant to external disturbances.
Loss to outdoor sunlight.
The remote control cannot be used in locations exposed to direct sunlight. ·Try using a light or fluorescent light.
What will happen if I use it at the same time as a TV remote control? ·Infrared remote control cannot see with the naked eye whether the element is lit or off, but it can be used with digital cameras and mobile phones.
If you look through the camera on the obi, you can see that the elements are shining.

◇ Let's try it2.6

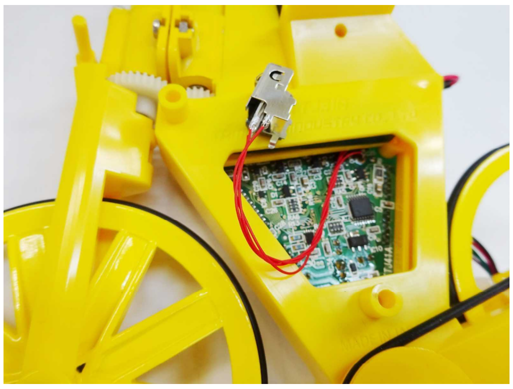


Photo 2.14 Light receiving module added

I tried connecting the remote control light receiving modules in parallel.
Lol.
The output of the light receiving module is an open collector. Because of this, multiple modules can be connected in parallel (OR), making it possible to operate from all directions.
Current automatic posture control bicycles have a light-receiving monitor pointing toward the rear.
One joule is installed, so the photo2.14 If you add a module toward the front like this, you can maneuver from almost all directions. The additional module in the photo is attached Photo 2.14 Addition of light receiving module to the vehicle body with double-sided tape.

2.7 three terminal regulator

2.7.1 What is a three-terminal regulator

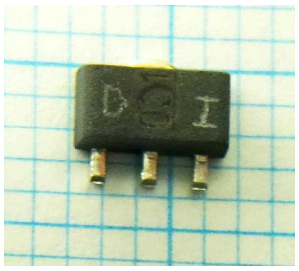


Photo 2.15 Three-terminal regulator

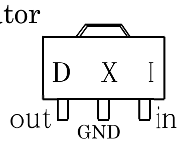
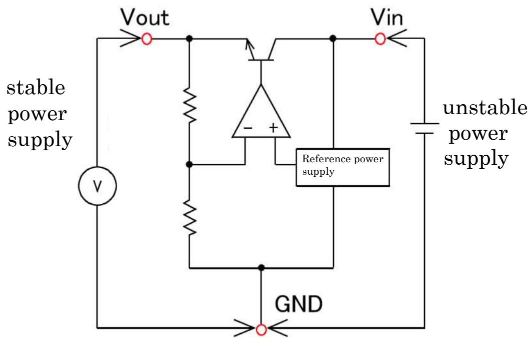


Figure 2.27 Dimensions regulator

As shown in Figure 2.28, the three-terminal regulator has three connection terminals (out, GND, in) as shown in Photo 2.15 and Figure 2.27, and as shown in Figure 2.28, it is a series regulator that inputs an unstable power supply, compares the reference voltage with the output voltage, and controls the voltage so that the output voltage is stable at a constant voltage.

Output voltages such as 1.8V, 2V, 2.5V, 3V, 18V, and 24 are available. For this bicycle, we use the three-terminal regulator TA48L03F shown in Photo 2.15 to create a control power supply of 3V from 4 AAA batteries (6V).



2.7.2 How to use a three-terminal regulator

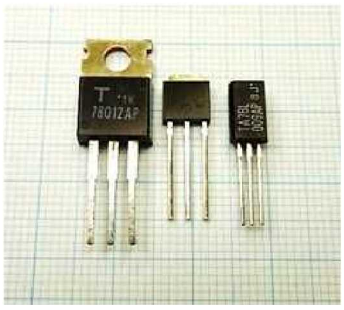


Photo 2.16Dip type three terminal regulator

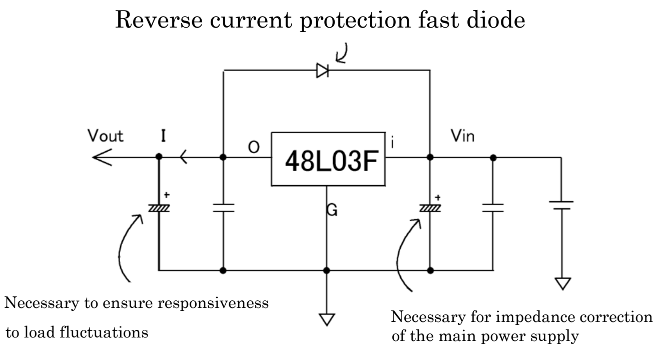

Reverse current protection fast diode
Vout
I
O
48L03F
i
Vin
G
Necessary to ensure responsiveness to load fluctuations
Necessary for impedance correction of the main power supply
Figure 2.29 How to use a three-terminal regulatorregulator

Photo 2.16 shows how to use a DIP-type three-terminal regulator, and Figure 2.29 shows how to use a general-purpose three-terminal regulator. Since the three-terminal regulator is a series regulator, pay attention to the voltage difference between the input Vin and the output Vout and the heat generated by the element.
The minimum input/output voltage differential, Vin-Vout, must be 1.7 V for general-purpose regulators and 0.5 V or higher for low-drop regulators. However, since the calorific value P of the element is the equation 2.5, which is obtained by multiplying this input/output voltage difference by the circuit current I, the design should take into account the balance between the voltage difference and the calorific value.

P= I (Vin－Vout) 〔W〕 ･･･formula 2.5

2.7.3 How to use a three-terminal regulator (advanced version)

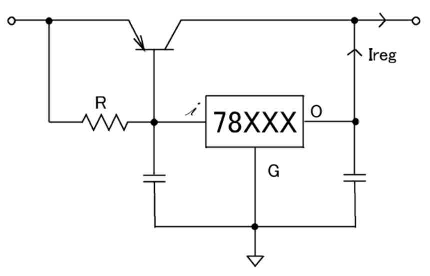


Figure 2.30 Current boost

In order to increase the current drawn through the current boost three-terminal regulator, the external transitor TR and the three-terminal regulator are circuited like Darlington's connections, as shown in Figure 2.30.

If the value of Ireg×R exceeds the VBE of TR, the output current I flows, so

$Ireg \times R = V_{BE}$ ・・・formula 2.5
$Ireg \times h_{FE} > I$・・・formula 2.6

$\dfrac{V_{BE}}{\dfrac{I}{h_{FE}}} > R$ ・・・formula 2.7

This current boost output current is said to be limited to about 5 ～ 6 times that of the three-terminal regulator used, so if the TR is 50 ～60 for hFE, the R is 5 ～6 Ω.

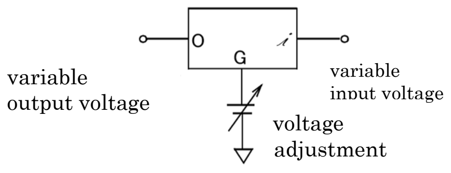


Figure 2.31 Variable output

A voltage adjustable three-terminal regulator can easily be used as a power supply with a variable output voltage by changing the potential of the G terminal, as shown in Figure 2.31. However, since the voltage difference between the input and output is the variable voltage range + the minimum input/output voltage, the amount of heat generated by the element indicated by the circuit current × the input/output voltage difference increases, and design needs to be required.

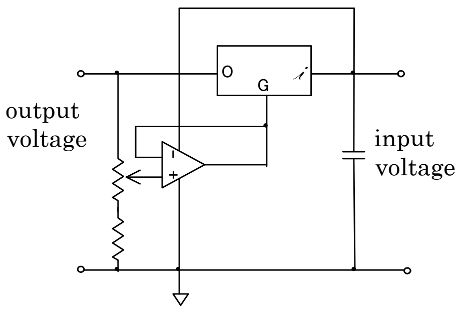


Figure 2.32 Voltage variable circuit side voltage

Figure 2.32 shows an example of a circuit for an adjustable output voltage power supply using a three-terminal regulator. The G terminal of the three-terminal regulator uses an operational amplifier to set the potential at low impedance. By supplying the op amp power supply from the input side, the range of adjustment of the output voltage setting is extended.

2.8 About A/D/D/A conversion

The bike uses a resistive string digital-to-analog conversion (AD5611) and a successive approximation analog-to-digital conversion (with built-in CPU) for the angular rate sensor input in the drift correction circuitry. Because analog-to-digital and digital-to-analog conversions are inverted and circuitically similar, the related A/D and D/A conversions are described in pairs here.

2.8.1 What is A/D/D/A conversion

(1) A/D conversion: Converts analog values such as voltage into digital values (bit weights). (2) D/A conversion: Converts digital values (bit weights) into analog values such as voltages. For example, if you convert an analog value of 6.5 [V] with an analog input range of 0 [V] ～ 10 [V] to a digital value with an A/D converter with 8-bit resolution = 256 (0 ～255), the digital value x is

If this is expressed by the weight of 8 bits, it is 128 + 32 + 4 + 1 = 165.

128	64	32	16	8	4	2	1
○	×	○	×	×	○	×	○

can be →○

none →×

The binary representation is 10100101b and the hexa representation is A5h.

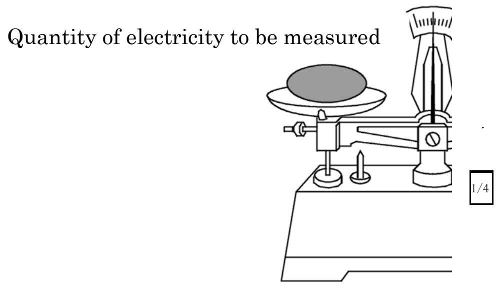
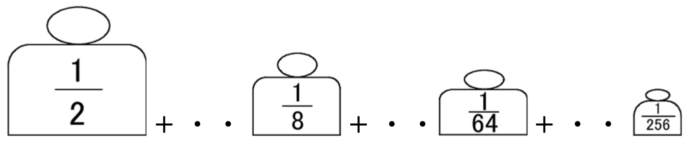


Figure 2.33 Balance scale

The analog-to-digital transformation of the sequential conversion method is like a balance in Figure 2.33. Find a combination of weights that balance by switching weights in order: 1/2 weight, 1/4 weight, and so on. The 8-bit digital value A5h described above is converted to digital to analog at 10 (V) full scale.

The D/A conversion is the sum of the weights of the bit weights shown in Figure 2.34.



＝165/255×10〔V〕

＝6.445〔V〕

Figure 2.34 Bit weight weight

bit weight $\left(\frac{1}{2}+\frac{1}{8}+\frac{1}{64}+\frac{1}{256}\right)\times 10$ 〔V〕 =6.445 〔V〕

In this way, the action of searching for the weighted combinations (bit weights) that are all balances is called A/D conversion, and conversely, the action of outputting an analog quantity that matches the weights combination (bit weights) is called D/A conversion. Note: The reason why the answer is not 6.5V is the full-scale error described below.

2.8.2 Specific example of A/D/D/A conversion part 1

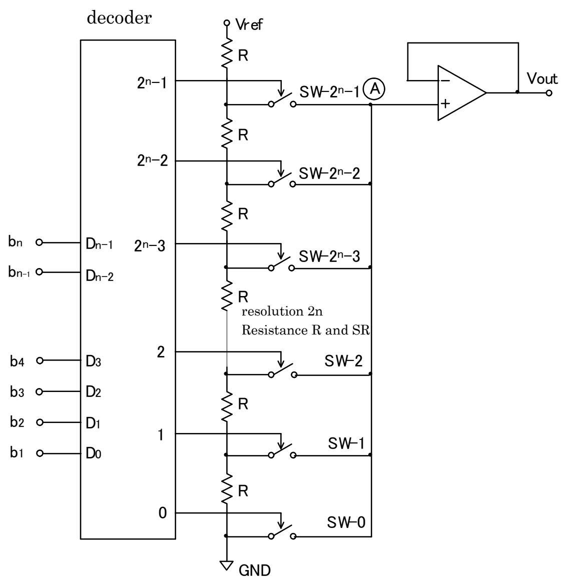


Figure 2.35 Resistor string method D/A conversion

Figure 2.35 shows a D/A conversion diagram using the resistor string method.

For this D/A conversion, as many resistors R of the same value as the resolution are connected in series, and as many output switches SW as the resolution are prepared at the connection points.

Only one output switch specified by the binary/decimal decoder is turned on, and the voltage divided by the series resistor is output.

The resistor string method has a simple configuration and is advantageous in linearity and conversion speed.

Also, by using the Vref, GND, and A connections in Figure 2.35, it can be used as a potentiometer or electronic volume as shown in Figure 2.36.

However, because the number of resistors and switches required is equal to the number of resolutions, the higher the resolution, the larger the circuit becomes and the more difficult it becomes to manufacture. (The normal resolution is 8~10 bits)

The maximum number of bits in the resistor string is 2^n-1, so a full-scale error of Vref-1LSB will occur, but it can be rounded off by adjusting the span.

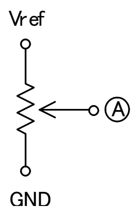


Figure 2.36 Potentiometer

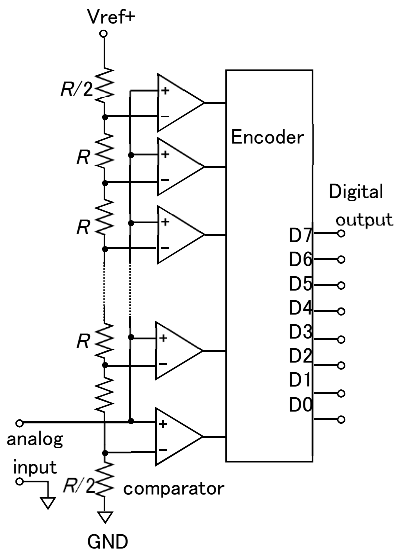


Figure 2.37 Flash type A/D conversion

(2) Flash-type A/D conversion The flash-type A/D conversion shown in Figure 2.37 is similar to the resistor-string digital-to-analog conversion, with the same number of series resistors and comparators as the resolution, and the voltage level of the analog input value and the string resistor are compared. Since the comparator up to the same voltage level as the analog input value is set to 'ON', high-speed analog-to-digital conversion is performed by encoding (encoding) the number of comparators in the 'ON' state and outputting it digitally. However, higher resolutions increase circuit size and power consumption, which is not very realistic. Normally, the resolution is up to about 10 bits, and it is used for image processing and other purposes by utilizing high-speed conversion capabilities.

2.8.3 Specific example of A/D/D/A conversion part 2

(1) R-2R Ladder D/A Conversion The resistor-string D/A conversion described in Section (1) above requires a power of 2 (e.g., 65536 sets of resistors and switches in the case of 16 bits) that is equal to the number of resolution bits, and is difficult to fabricate at high resolution. Therefore, the shape of the rudder resistor is devised to reduce the number of resistors and switches, resulting in the D/A conversion of the R-2R ladder method.

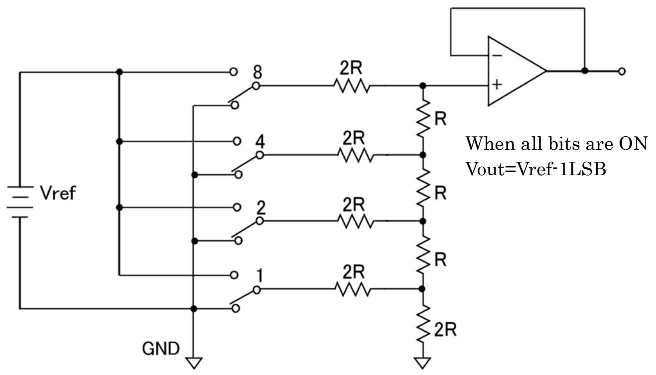


Figure 2.38 Voltage addition type R-2R ladder method D/A conversion

We will use the R-2R ladder method D/A conversion to explain how D/A conversion works. In the R-2R ladder method of voltage summing type D/A variant + conversion, when switch 8 on the top side is connected to the Vref side, half of the voltage of Vref is added to the output, and at the next switch 4, half of the Vref half (1/4) is added, and half of the voltage is added each time the bit goes down.

If the number of bits is infinite, Vout=Vref, but in reality, a full-scale error occurs, and Vout = Vref-1LSB, which is spanned by the output amplifier. The R-2R ladder can be easily analyzed using the superposition theorem or the Feng-Thevenin theorem. Compared to the string method, the R-2R ladder method D/A conversion simplifies the circuit but reduces linearity due to resistance errors. In particular, the resistance error is an important factor when the number of conversion bits increases.

(2) Successive approximation method A/D conversion

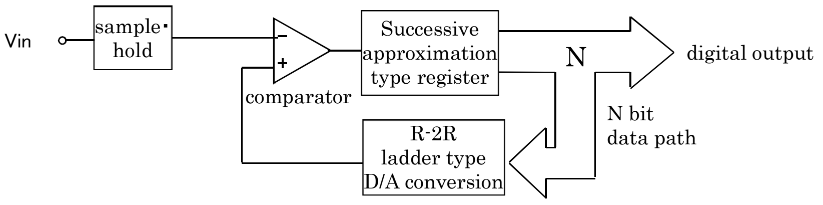

Vin
sample・hold
comparator
Successive approximation type register
N
digital output
N bit data path
R-2R ladder type D/A conversion

Figure 2.39 Successive approximation method A/D conversion

Successive approximation analog-to-digital conversion works in the same way as weighing weighing. Using the balance in Section 2.8.1 above as an example, the order in which the A/D transformation is performed is explained. (1) Since successive approximation A/D conversion cannot be A/D conversion in an instant like the flash type, the analog input value at the start of conversion is maintained by a Mr./Ms. pull hold circuit so that the analog input value does not change during A/D conversion. This is the state in which the amount of electricity to be measured is placed on the balance plate. (2) Place 1/2 full-scale weight on a weight dish. If the weight side is light→ now with the weight on it, and the weight side going to (3) is heavy, → put the weight on it now, put it down and go to (3) and put the 1/4 weight of the full scale on the weight plate. → the weight side is light, → the weight side going to (4) is heavy with the weight on it now, put the weight on it now, and go to (4) and do the same work sequentially with the weight going to (4) 1/8, 1/16, 1/32・・・・・ and so on. (5) Repeat the same process until the smallest weight, and finally the weight on the weight dish becomes the weight of the digitally converted bit. Return the Mr./Ms. pull hold to the Mr./Ms. state. In Fig. 2.39, the R-2R ladder type D/A conversion is a weight, the comparator is a balance, and the successive approximation register is a weight plate, and the above (1)~(5) is automatically executed.

In voltage-comparison analog-to-digital conversions, analog-to-digital conversions that require conversion time, such as successive approximation conversions, use a Mr./Ms. pull-and-hold circuit to prevent the analog value from changing during conversion.

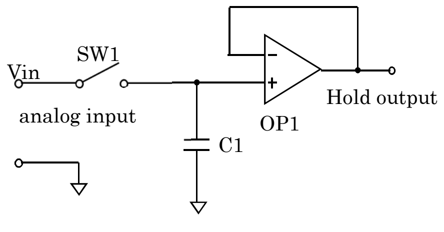


Figure 2-C Sample hold

Figure 2-C shows the basic shape of the Mr./Ms. pull and hold circuit. Op amp OP1 in Figure 2-C selects the lowest input offset current. The capacitor C1 for the Mr./Ms. pull hold is a capacitance on the order of pF. After SW1 is turned off, the analog value of Vin stored in C1 is retained, resulting in a Mr./Ms. hold. In a real circuit, the leakage current of SW1, the natural discharge of C1, and the offset current of OP1 determine the performance of Mr./Ms. pull hold.

Column 2.4 Multiplexer

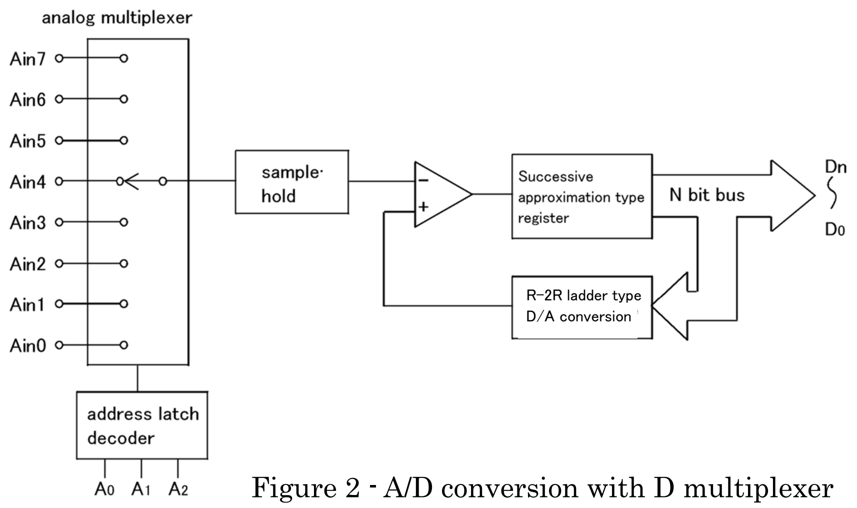


Figure 2 - A/D conversion with D multiplexer

Figure 2-D is an A/D converter with an analog multiplexer (analog signal switch). Switch multiple analog inputs and perform analog-to-digital conversion in sequence. Modern CPUs have built-in A/D conversion and D/A conversion, so there is no need to be aware of the hardware, but a circuit like the one shown in Figure 2-D is built into the CPU.

The multiplexer can specify individual channels or scan specifications, but it is inevitably affected by the previous conversion channel.

If the converted value of the previous channel is large, an error will appear in the larger converted value of the next channel. This is likely due to the charge accumulated between the multiplexer and the sample and hold. Countermeasures against this inter-channel interference include: (1) channel placement taking into account signal priority; and (2) connecting to GND without using the channel immediately before the important channel.

◇Try it out

2.7 Figure 2.40 shows a test circuit for a simplified successive approximation analog-to-digital conversion that combines a 4-bit R-2R ladder digital-to-analog conversion with a comparator. Build it on a breadboard or a universal board. The circuit configuration is explained. 1/2 of OP1 and OP2 convert a resistor-divider circuit (high impedance) to low impedance using followers. 2/2 of OP2 is a comparator that compares the analog input voltage to the D/A output of R-2R. When the analog input voltage < D/A output voltage, LED1 lights up. R5 is a positive feedback resistor that provides a 10 kΩ/1 MΩ hysteresis range for the comparator. The op amp TB75358 used can operate from a single supply, but the ± 12V power supply is available. R10~R17 and SW1~SW8 constitute a 4-bit R-2R ladder D/A conversion.

How to use:
Set the Vref. Fig. 2.40 Turn VR2 in the A/D conversion and D/A conversion test circuit and adjust the voltage between CH2 and GND so that it is 10 [V]. This voltage is the full-scale value for the analog-to-digital and digital-to-analog conversions. SW8 ~ SW1 to the ↓'0' side.

[4-bit D/A conversion]

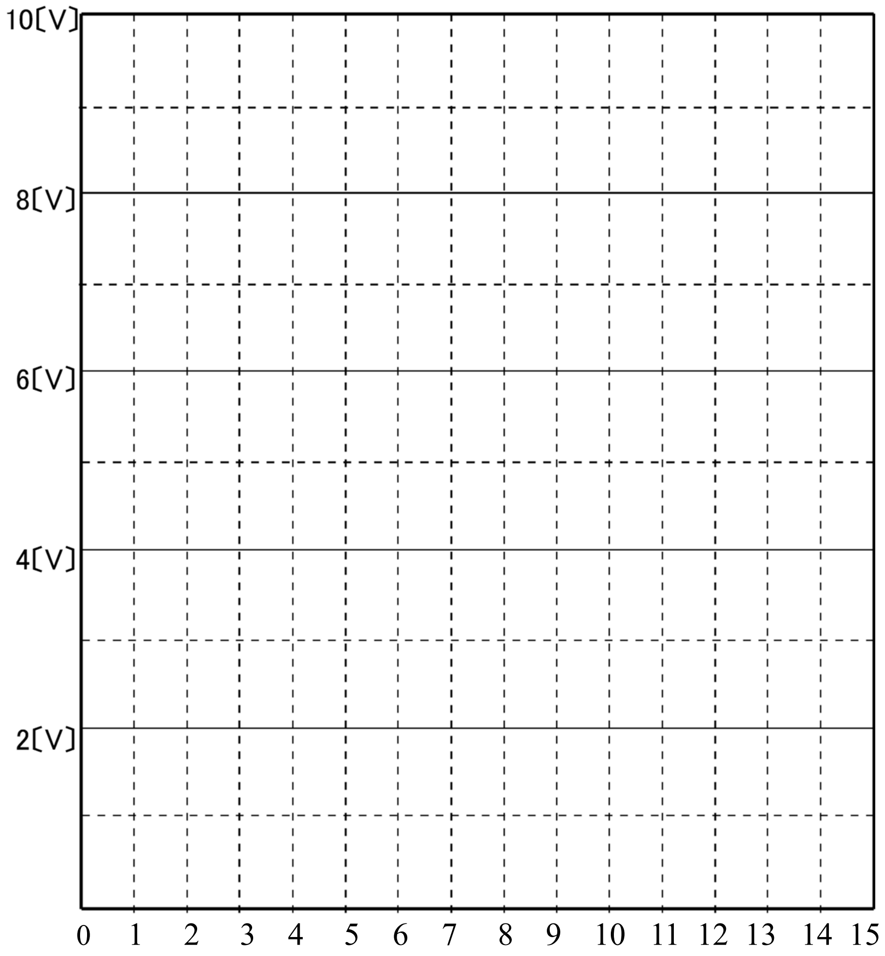
(1) Measure the voltage between CH4 and GND when SW8 ~ SW1 are all downward. 〔SW-0〕

(2)Measure the voltage between CH4 and GND when SW8 ~ SW1 is down, down, down, and up. 〔SW-1〕

(3)Measure the voltage between CH4 and GND when SW8 ~ SW1 is down, down, down, down, and down. 〔SW-2〕 (

4)Repeat in this order until [SW-15]. (5) Record in the table below and plot on the graph.

SW	CH4〔V〕
15	
14	
13	
12	
11	
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	
0	

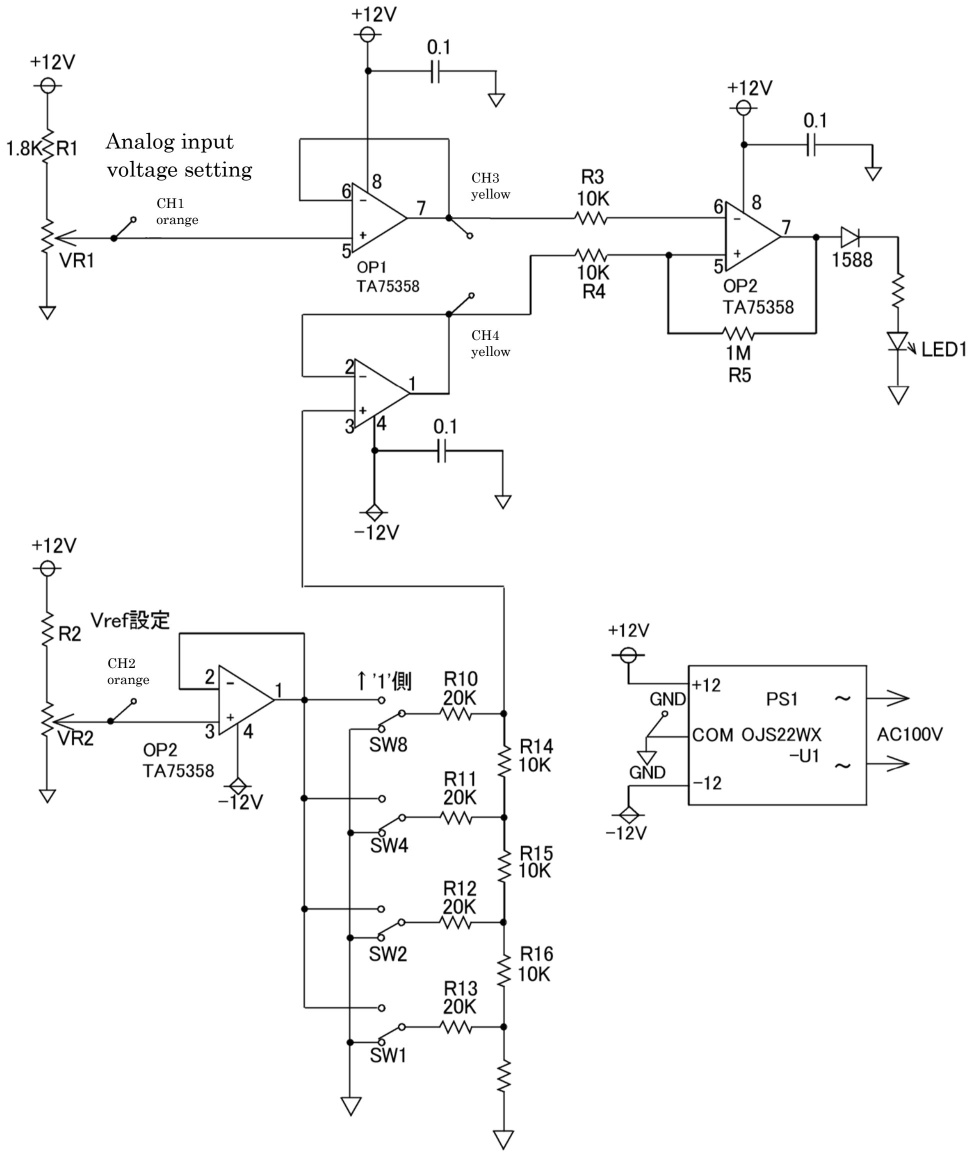


Have you confirmed that the analog output value changes by 1 LSB due to the combination of SW8 ~ SW1

SW8 ~ SW1 is a 4-bit D/A converter when operated by a computer. Next, let's use the 4-bit R-2R ladder digital-to-analog transform to try out the successive approximation analog-to-digital transformation shown in Figure 2.39.

[4-bit A/D conversion]

① Set the analog input voltage Vin. Rotate VR1 to set the analog input voltage to any value. (About 7V is easy to understand)
②"Turn SW8 'ON'." This is the same as ② "Place 1/2 full scale weight" in item (2) of 2.8.3 above.
If LED1 does not light up, the weight side is in the same state as light, so go to (3) with SW8 ON (as it is with the weight you just put on it). When LED1 lights up, the weight side is in the same state as heavy, so SW8 is 'OFF' (lower the weight you just put on it) and go to (3).10K



③ "Turn SW4 'ON'." This is the same as "Place a weight of 1/4 of the full scale."
LED1 does not light up = the weight is light → SW4 goes directly to ④
LED1 lights up = weight is heavy → SW4 is 'OFF' and goes to ④

④ Execute "Perform the same work sequentially with weights 1/8, 1/16, etc."
Perform the same work as ③ above in the order of SW2:SW1.

⑤The output value of analog/digital conversion is
The sum of the weights of the switches SW that are 'ON' x Vref.
For example, when SW8 and SW1 are 'ON' and Vref is set to 10 [V],
Digital conversion value = (1/2 + 1/16) × 10 [V] = 9/16 × 10 [V] = 5.625 [V] Compare with the voltage between analog input voltage CH1 and GND.
The converted digital value is about 1 LSB smaller at most. This is the full-scale error.
Turn VR1 a little to change the analog input value a little, and repeat steps 1 to 5 to check the A/D conversion operation.

Chapter 3 Core CPU for Embedded Automatic Attitude Control

The CPU of the bicycle can be selected from Toshiba Corporation's TLCS-900 type or ARM type. The optical remote control is only TLCS-900 type. Table 3.1 summarizes both CPUs. Both CPUs are designed for small-scale embedded applications and have the same package, operating voltage, and integrated I/O and processing power. TLCS-900 is a CPU classified as CISC type and ARM is classified as RISC type.

Core CPU	Model name	specification	package

TLCS−900 /L1	TMP91FW27UG
	Toshiba
		ROM=128kbyte
		RAM=12kbyte
		clock=27MHz
			LQFP64−P−1010−0.50D

ARM
Cortex−M3
	TMPM332FWUG
	Toshiba
		ROM=128kbyte
		RAM=8kbyte
		clock=40MHz
			LQFP64−P−1010−0.50E

Table 3.1 CPU overview

3.1 TLCS-900 Architecture

TLCS-900 is a generic name for Toshiba's original 16/32-bit CISC core CPUs. Although there are multiple CPU cores, all types have the same set of registers, which are fully 32-bit, and can use the same compiler assembler, as well as the ability to repurpose source programs from assembly language descriptions. In addition, the TLCS-900 distinguishes between 16 bits and 32 bits at the end of L1 and H1, and although there are differences in the ALU, internal bus width, and the number of stages of the pipeline, the usage including I/O is the same, so from the user's point of view, the TLCS-900 is a type of core. Processing power is available up to 80 MIPS with 32-bit CISC instructions. The TLCS-900, the predecessor of the TLCS-900, was designed to be Z80 upcompatible, and the TLCS-900 is an extended version of the Z80 in register names, etc., so the Z80 program in the assembly language description can be used with some modifications.

3.1.1 Features of the TLCS-900

The TLCS-900 is designed with a thorough CISC philosophy to improve performance, as shown below.

①Generated code is short

A computer is a machine that reads (fetches) and executes instructions (codes) in memory. Instructions captured in a computer execute pipeline processing, and generally one instruction is processed at one clock, but since fetching involves instructions and operands, some things are not completed in one clock, and the processing power of the computer = fetch speed. In other words, the shorter the fetch time, the faster the computer processing speed will be, so the TLCS-900 uses variable-length instructions and mixes operands in the instruction word to reduce the code so thoroughly that it becomes difficult to disassemble it, thereby improving performance.

② Good addressing orthogonality
Addressing is a method of indicating the data specified by an operand or the storage location of the data. A computer executes a program written using a combination of instructions and addressing as shown in the following format.

```
unsignedintMEM,a;        \   C language description:
a   +=   MEM;            |  Addition of MEM and a
|    |    └ operand      |
|    └── order           /
└── operand

ADD   XBC,(MEM)             ; Assembly description:
 |     |     └ 2nd
 |     |       operand      ; Add 4 bytes from the memory address specified
 |     └── 1st operand      ; by  (MEM) to the XBC register.
 └── add order
```

CPUs that are less limited in this combination of instruction and addressing are said to have good addressing orthogonality. All registers, memory, and stacks can be specified for addressing. Even if the processing speed expressed in MIPS is the same between CPUs and CPUs that are limited to only registers, such as RISC type, the actual execution speed of the program will be several times different.
The TLCS-900 is a CPU with excellent orthogonality of addressing.

③Function to automatically generate an appropriate operand size
TLCS-900 has no jump width restrictions such as segmentation for branch instructions such as jump instructions and call instructions, and there is no overhead such as always using operands with the maximum address width.
The compiler, assembler, and linker, which are language tools, can determine the jump width when reading the source program, so the TLCS-900 language tool automatically calculates the branch width and selects the appropriate 8-bit, 16-bit, or 24-bit branch width. It uses an innovative technique to generate width operands.

④Conditional CALL command/Conditional RET command
There are conditional instructions for subroutine calls and returns from subroutines.
On CPUs where conditional CALL cc and RET cc cannot be used, CALL cc is a combination of Bcc + JSR, and RET cc is a combination of Bcc + RTS, which will definitely slow down processing by increasing the number of instructions by one.

⑤ Rich in CISC instructions
There are a variety of instructions unique to CISC, such as MIRR (mirror) instructions, DAA (decimal correction) instructions, and powerful bit instructions, and there are no restrictions on operand size, and 8-bit, 16-bit, and 32-bit widths can be combined, so the result The program code is shortened and can be executed at high speed.

As a slight aside

low-priced, high-performance CPUs like the Raspberry Pi (a bit different from embedded CPUs) are now easily available, so there is no need to be aware of the CPU architecture, but embedded CPUs Back when CISC was at its peak, the architectures of the TLCS-900 and M16/32 (Renesas/Mitsubishi) were two of the best.
As microcomputer engineers, we poured our energy into understanding his CPU architecture, and we were very impressed.

3.1.2 Register configuration of TLCS-900
As shown in Figure 3.1, the register configuration of the TLCS-900 is fully 32 bits, while the register name is upward compatible with the Z80, making it very easy to use because all registers can be used as an accurator. The register configuration is unified for all types of CPU cores of the TLCS-900, and programs can be used. The following is a brief description of the registers.

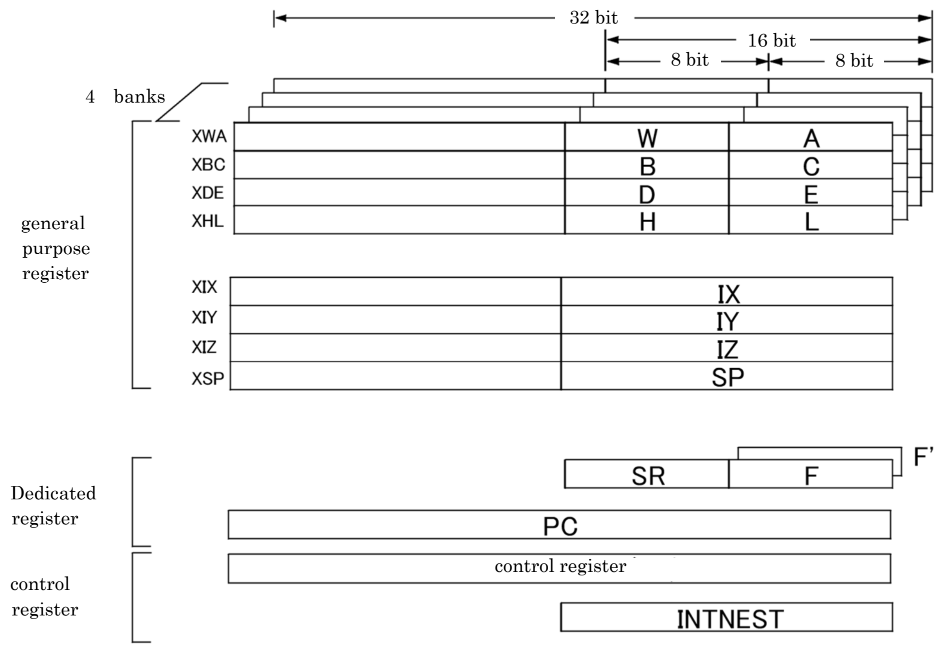


Figure 3.1 TLCS-900 register configuration

(1)General-purpose register
There are seven 32-bit general-purpose registers, XWA to XIZ, shown in Figure 3.1, which can be used as accumulators or index registers.

There are four banks of the four XWA to XHL with the same configuration, and bank switching can be executed with one instruction, so registers can be saved at high speed during interrupts.The number of general-purpose registers may seem small compared to RISC-type CPUs of the same class, but this is because compared to RISC-type CPUs, where all calcula- tions can only be performed in registers, CISC type CPUs, especially the TLCS-900, use the first operand as the first operand. This is because addressing is powerful, such as being able to specify a memory area on the (destination) side, so there is no need for many general-purpose registers.
General-purpose registers can be specified as 8-bit, 16-bit, or 32-bit wide.Figure 3.2 shows a specific example of width specification using the BC register as an example.8 -bit specification: B register, C register, QB register, QC register 16-bit specification : BC register/QBC register 32-bit specification: XBC register
Note: QB, QC, and QBC are extended instructions and are 1 byte longer.

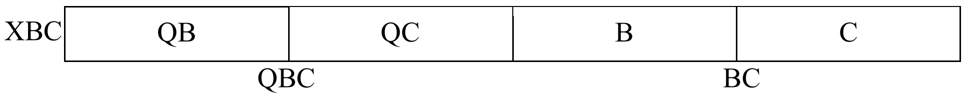


Figure 3.2 Register width specification

(2) Stack pointer (XSP)
A 32-bit wide register that points to the stack location when operating memory as a stack. The stack is used to store the return address of subroutine calls, and is used as a stack frame as a method for passing arguments in the C language.

(3) Status register/flag register (SR/F)
The upper byte of the 16-bit wide register is the status register SR, and the lower byte is the flag register F.
Figure 3.3 shows the bit arrangement of status register SR. IFF2 to IFF0 are interrupt mask registers that indicate the interrupt levels that the CPU can currently accept.

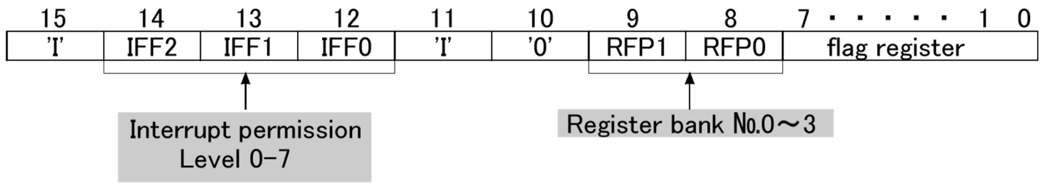


Figure 3.3 Status register(SR)

When the value of IFF2 to IFF0 is 1 or less, all interrupts are enabled, and when the value of IFF2 to IFF0 is 7, interrupts are disabled. After the CPU is reset, IF F2 to IFF0 are initialized to 7, and when the interrupt enable instruction EI n is executed, this value becomes n, allowing interrupts up to interrupt request level = n.

Register file pointers RFP1 and RFP0 specify No. 0 to 3 of the four register banks. The default value of register bank No. is 0 bank.
For interrupts that require speed to save registers, use an instruction such as LDF n in the first line of the interrupt service routine to specify the register bank number to be used during the interrupt. To return from an interrupt, use the RETI instruction to return to the state before the interrupt.
In the flag register F shown in Figure 3.4, the corresponding flag changes and is stored according to the CPU's calculation results, and the flag is automatically referenced by the judgment instruction in the program, rather than being directly checked by the programmer.

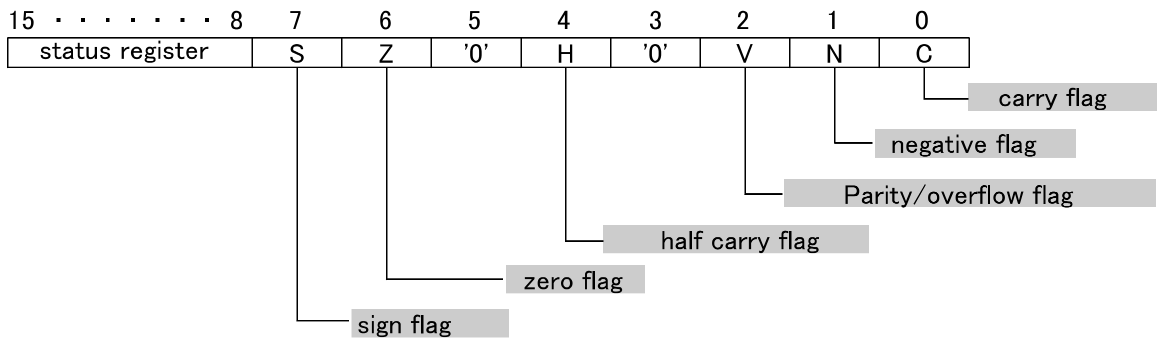


Figure 3.4 Flag register (F)

(4) Program counter (PC)
The TLCS-900 uses the lower 24 bits of the 32-bit wide program counter PC to specify a 16 MB memory area. The program counter contains the ``address of the drawer'' that will be opened next after the ``memory drawer containing the program''. Normally, the program counter value is incremented sequentially, but if there is a jump or call instruction in the program, the jump address value specified there is copied to the program counter.

(5) Control register (CR)

The control register includes a configuration register for DMA transfers and a nesting counter for interrupts, and is treated like an intermediate between a register and an I/O and accessed with an LDC instruction.

The TLCS-900 has 4 or 8 DMA transfer channels as shown in Figure 3.5, consisting of the following 4 setting registers.

DMA transfer is an automatic transfer function in which the CPU does not issue transfer addresses, but the DMA controller manages transfer addresses and transfer counters, and combines this with software interrupts.

DMASn : 32-bit DMA source address register

DMADn : 32-bit DMA destination address register

DMACn : 16-bit DMA transfer counter

DMAMn : 8-bit DMA transfer mode register
(The n at the end indicates the channel number)

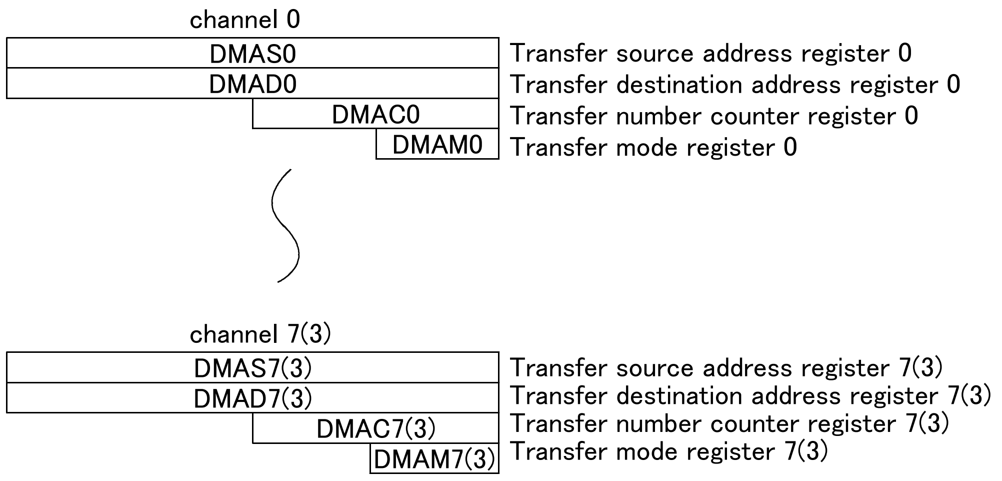

channel 0
DMAS0
DMAD0
DMAC0
DMAM0
Transfer source address register 0
Transfer destination address register 0
Transfer number counter register 0
Transfer mode register 0
channel 7(3)
DMAS7(3)
DMAD7(3)
DMAC7(3)
DMAM7(3)
Transfer source address register 7(3)
Transfer destination address register 7(3)
Transfer number counter register 7(3)
Transfer mode register 7(3)

Figure 3.5 DMA Control Register

(6) Bank register

The general-purpose registers XWA to XHL shown in Figure 3.1 are provided in four banks using a bank switching system. Bank switching can be performed by using the LFD n instruction to rewrite the status registers RFP1 and RFP0 in one clock, so registers can be saved faster than using the PUSH and POP instructions when an interrupt occurs.

3.1.3 TLCS-900 interrupt

Interrupt control of the TLCS-900 shown in Figure 3.6 is divided into interrupt acceptance processing performed by the interrupt controller and interrupt service routine processing performed by the CPU.

The interrupt controller and CPU are connected by an interrupt request signal INTRQ and an interrupt vector representing the interrupt factor.

Interrupt processing is performed in the following order.

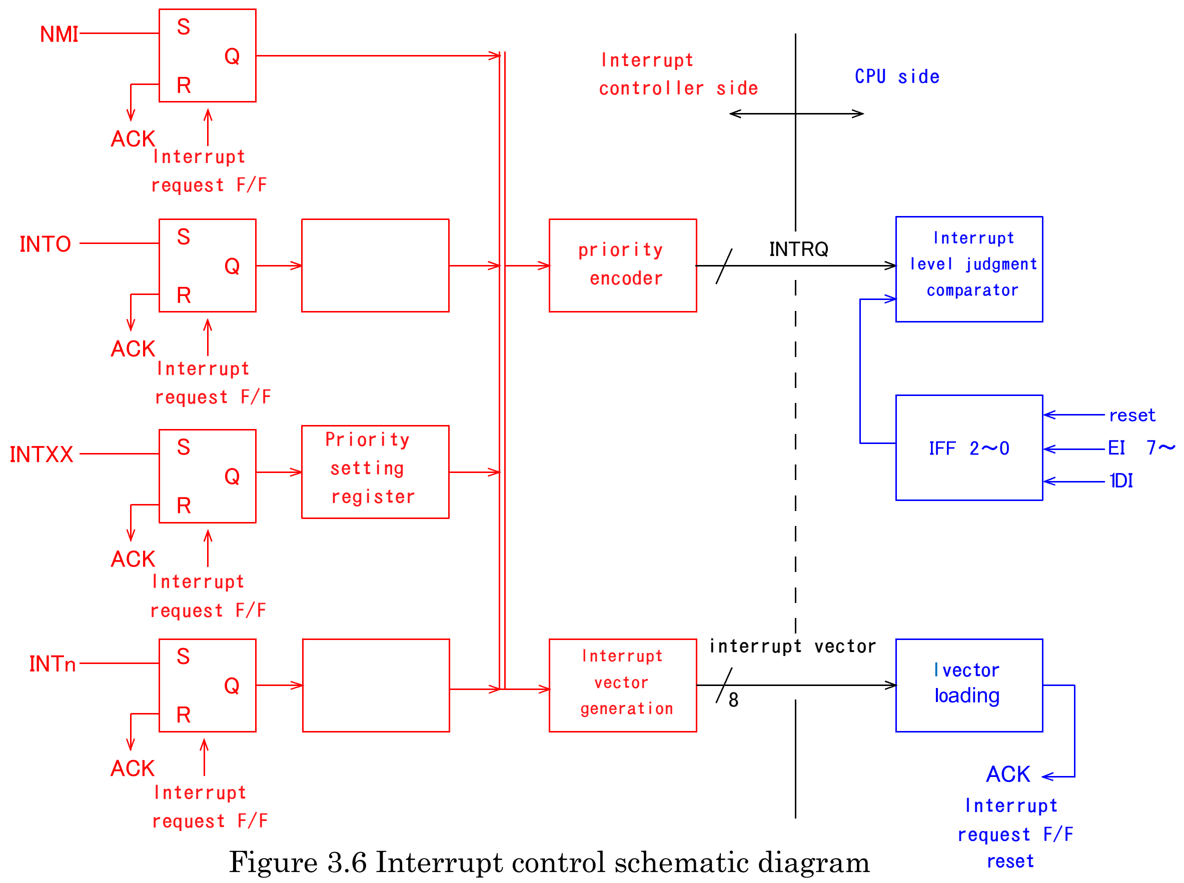
① When an interrupt occurs, set the interrupt request F/F for each cause.

②The interrupt controller sends the interrupt request level and interrupt vector to the CPU according to the preset interrupt request level order for each interrupt source.

③ The CPU side receives the interrupt request level, compares it with the current value of the interrupt mask register IFF, and accepts the interrupt if the interrupt request level is greater than or equal to the value of IFF.

④ The CPU reads the interrupt vector and at the same time returns a vector read ACK to the interrupt controller.

⑤When the interrupt controller receives the vector read ACK from the CPU, it resets the corresponding interrupt request F/F.



NMI
S
Q
R
ACK
Interrupt request F/F
INT0
INTXX
Priority setting register
INTn
Interrupt controller side
CPU side
priority encoder
INTRQ
Interrupt level judgment comparator
IFF 2~0
reset
EI 7~
DI
Interrupt vector generation
interrupt vector
8
I vector loading
ACK
Interrupt request F/F reset

Figure 3.6 Interrupt control schematic diagram

⑥The CPU PUSHes the program counter PC and status register SR. Writes the value of the accepted interrupt level +1 to the interrupt mask register IFF. Increments the value of the interrupt nesting counter INTNEST by 1.

⑦The CPU jumps to the address indicated by the value (0FFFF00H + interrupt vector) address data in the vector table, executes the interrupt service routine, and with the RETI instruction after the interrupt ends, pops the status register SR and program counter PC, and loads the interrupt nesting counter. Decrease the value of INTNEST by -1.

Column 3.1 Interrupts

Interrupts are also subroutine calls in a broad sense. The difference between an interrupt and a subroutine call is in the method of specifying the jump destination. In the case of a subroutine call, the jump destination address is specified in the program like CALL label, but since an interrupt is an event that starts from a sudden interrupt request event from I/O, there is no place to write the jump destination. Therefore, a number (interrupt vector) is assigned in advance to the I/O that requests an interrupt, and a memory area (interrupt vector table) is prepared in which the jump destination address corresponding to that number is written. When an interrupt occurs, the CPU executes the jump destination interrupt service routine written in the vector table corresponding to the vector number sent from the interrupt controller.

Column 3.2 20MIPS wall

As shown in Figure 3-A, a microcomputer consists of a core CPU and memory (ROM/RAM)/I, each of which is connected by a bus (address bus, data bus, control bus, etc.). As core CPU speeds increase, if the speed difference between I/O and CPU increases, it will no longer be possible to connect with a direct bus. When MIPS (the number of instructions that can be executed per second), which is one of the computer performance indicators, is 20 MIPS or more, the I/O/memory access time is about 50 ns, which is the limit of direct bus/direct fetch. If the speed is higher than this, cache memory is used to run only the core CPU peripherals at high speed, and a ``bridge'' is used to access the low- speed bus.

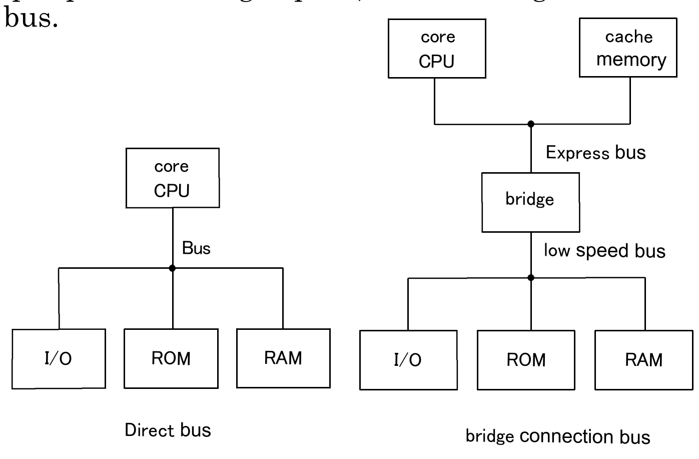


Figure 3-A Microcomputer bus configuration

The ARM architecture also uses this bridge to connect the AHB (high speed) bus and the APB (low speed) bus.

Bus configuration is also a deciding factor when selecting an embedded CPU.

3.2 ARM Architecture 3.2.1 History of ARM Company Before explaining the ARM processor and ARM architecture, I will explain the history of ARM Company. ARM was established in 1990 as Advanced RISC

Machines Ltd., a joint venture between Apple Computer, Acorn Computer Group, and VLSI Technology. In 1998, the company changed its name to ArmHoldings when it went public. In 2016, the company was acquired by SoftBank and continues to operate to this day.

3.2.2 ARM's sales strategy ARM does not manufacture CPU processors, but rather sells the right to use the ARM architecture intellectual property (IP) license, and receives royalty income from manufacturers who receive a portion of the CPUs they sell. It is established as a company. Currently, there are over 1,550 companies using IP licenses, including IBM, Motorola, Nintendo, Sharp, and Samsung Electronics.

3.2.3 Sales methods of semiconductor manufacturers

Each semiconductor manufacturer receives the ARM architecture blueprint from ARM and incorporates it into their own CPUs.

Then, each company creates and sells its own CPU processor by adding the necessary memory, peripherals, input/output, etc., and other functions. This reduces development time and costs for the architecture within the CPU, while also making it possible to quickly sell new processors.

3.2.4 ARM architecture details

Figure 3.7 shows the ARM architecture details. I will explain in the order of the numbers in the diagram.

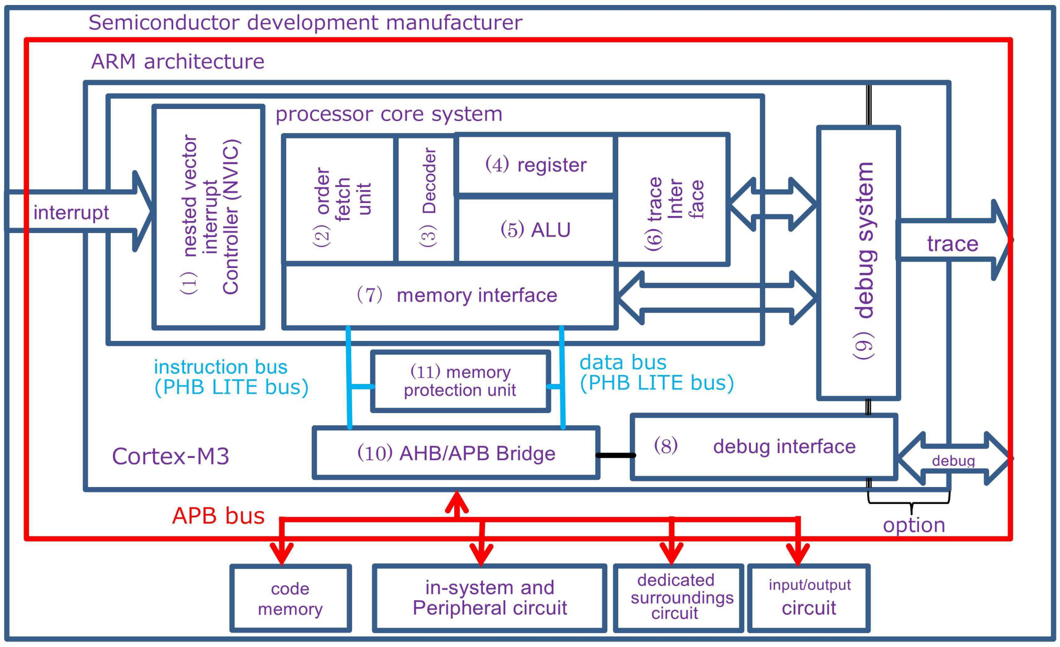


Figure 3.7 ARM architecture details

(1) Nested Vectored Interrupt Controller (NVIC) The nested vectored interrupt controller (NVIC) can be more clearly understood by dividing it into nested and vectored interrupts. The nested type is a function that appropriately manages the order of interrupts based on the priority of the interrupt when it occurs. When a vectored interrupt occurs, it refers to a place called a vector (interrupt vector table) where the names of interrupt sources are written, and sends the interrupt name to the CPU. External interrupts can be set between 1 and 240. Interrupt priority is set using an 8-bit register divided into two groups. In most actual products, priority can be set using 3 to 8 bits. When saving and restoring registers during tail chaining, preemption , late arrival, and three interrupts, continuous interrupt processing is possible without unnecessary processing. When an interrupt occurs, the NVIC handles the interrupt according to the priority of the interrupt, and at the same time performs a process called stacking to save the currently used registers.

The stacked registers are determined by the "ARM Architecture C/C++ Language Standard Procedure Call Conventions (AA PCS)" and include registers R0 to R3, R12, LR (link register), PC (program counter), and PSR. (Program status registers) are saved to the stack. In addition, the stack to be saved starts from the highest address of the static memory, and in the case of Cortex-M3, register values are saved sequentially from 0x3FFFFFFF to lower addresses.

Figure 3.8 shows the memory map of Cortex-M3.

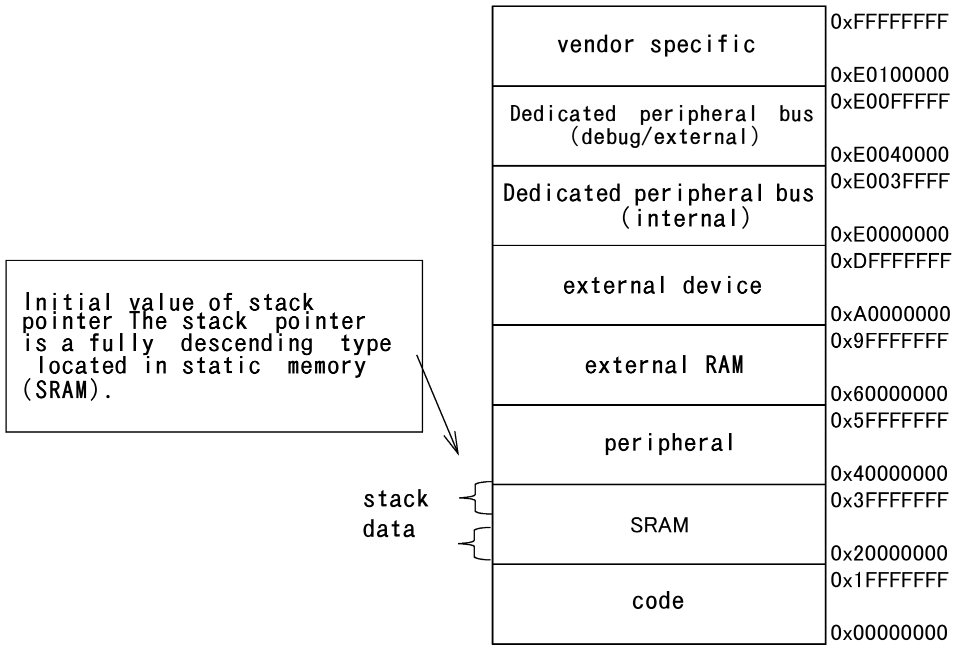


Figure 3.8 Cortex-M3 memory map example

(2) Instruction fetch unit The instruction fetch unit is a unit that reads the instruction program to be executed from memory. The read instruction program is passed to the decoder.

(3) Decoder
What is a decoder? It converts the instructions passed from the instruction fetch unit into machine language.

(4) Registers
Registers are storage devices within the CPU that temporarily store calculation results , etc. In the case of Cortex-M3, there are 13 32-bit general-purpose registers. Table 3.2 shows a list of registers.

register name		Functions (bank register, etc.)	
R0		general purpose register	lower register
R1		general purpose register	
R2		general purpose register	
R3		general purpose register	
R4		general purpose register	
R5		general purpose register	
R6		general purpose register	
R7		general purpose register	
R8		general purpose register	upper register
R9		general purpose register	
R10		general purpose register	
R11		general purpose register	
R12		general purpose register	
R13(MSP)	R13(PSP)	Main stack pointer (MSP) Process stack pointer (PSP)	
R14		Link register (LR)	
R15		Program counter (PC)	

table3.2General-purpose register list

lower registerR0fromR7is for all instructions that specify general-purpose registers (Cortex-M3With instructions available in16bit instructions and32bit instructions). upper registerR8fromR12specifies a general-purpose register32Can be accessed with bit instructions, but unlike lower registers16It cannot be accessed with bit instructions.R13(MSP,PSP)is called the main and process stack pointer, and stores the current stack pointer position.OSBasically, if you do not useR13teethMSP(main stack pointer). In addition to general-purpose registers, table3.3status registers likexPSR · There are interrupt mask registers, control registers, etc., which can be accessed by special instructions.CPUThis is a special register for control.

register name	function
xPSR (Program status register)	ALU flag (zero flag, carry flag) Contains the execution status and the number of the currently executing interrupt
PRIMASK (interrupt mask register)	Disable all interrupts except non-maskable interrupts (NMI) and HardFault
FAULTMASK (interrupt mask register)	Disable all non-NIM interrupts
BASEPRI (interrupt mask register)	Specified priority or low priority interrupts all prohibited
CONTROL (control register)	Sets privilege state and stack pointer selection

table3.3 Special register list

(5)ALU

ALUteethArithmetic Logic UnitIt is an abbreviation for ``arithmetic logic device'' in Japanese, and performs theoretical operations and four arithmetic operations. Cortex-M3teeth32bitCPUThereforeALUtoo32Processes bit by bit.

(6)trace interface
(7)memory interface
(8)debug interface
(9)debug system
aboveFourThe items are interfaces and systems related to debugging, which will be described later.Cortex- M3The debug system isCoreSightThe debug architecture covers a wide range of debug systems, including debug interface protocols, debug bus protocols, debug component control, security functions, and trace data interfaces. These components are typically used only by debugger software and not by your application.

(10) AHB/APBbridge
coreCPUSurrounding high-speed system busesAHBandI/OLow-speed veriferal buses such asAPBIt is a bridge that serves as a bridge. thisAHB/APBThe bridgeARMbus management architecture,Cortex-M0The same bridge is used in .

column3.3 About low power consumption

ARMThe instruction language architecture ofRISCarchitectureCISCarchitecture CPU(Intelof CoreIt can be said that the hardware size is smaller and power consumption tends to be lower than that of other models (e.g. series). Also, the instruction architecture isRISC However, we focused on code density.CICSWe are designing instructions close to . Therefore, by making full use of coding technology, he is able to extract performance that exceeds the operating clock of the processor. Specifically, when we compared the power consumption of Toshiba Corporation's equivalent class embedded CPUs, TLCS900H1, 900L1, and Cortex-M3, there was no big difference between them, and they consumed power in the order shown below, which work well.

performance high TLCS900H1 ＞Cortex-M3＞TLCS900L1 performance low
power consumption high Power consumption low

column3.4 CMSISabout

In the development and maintenance of systems using microcomputers, it is very advantageous to utilize technological assets accumulated in the past. ARMThe company is trying to improve the portability of its software.Cortex- MSoftware interface standard for series CMSIS(Cortex Microcontroller Software Interface Standard)announced. This allows peripheral settings andDSPLibrary · ROTThis improves the reusability of interfaces, debugger interfaces, etc., making development more efficient.Cortex-MThe series processor itself is also conscious of standardization.CMSISIt is also designed to accommodate differences in microcontrollers for easier software reuse. Also CMSISIt also has the advantage of making it easier to participate in the standard because it is a guideline and does not require certification.

3.3 About debugging

Bicycle that doesn't fallARMversion ofCPU TMPM332FWUG(Cortex-M3)has a debug interfaceCoreSightis built-in, which isARM Cortex-MThis is one of its major features. CPUIn- circuit emulator as speed increasesICENow that the practicality of CoreSightteethCPUWhile minimizing the burden onCPUControl, memory access , trace functions, etc.ICEIt has the same or better performance.
TMPM322FWUGhas a debug interfaceSWD (Serial Wire Debug)unit and trace output EMT(Embedded Trace Macrocell)andSWV(Serial Wire Viewer)unit is included.

3.3.1 SWD overview

SWD teeth ARM The company Cortex This is a debugging tool developed for Core Sightadopted in 2. It is a wire communication interface, JTAGcan be substituted for Debug control with bidirectional data signal (SWDIO)A clock synchronized with (SWCLK) of 2 do it with a book ARM This is a proprietary serial interface. PhysicallyI2C Semi-similar to bus communication2Heavy communication.

3.3.2Selecting a debug interface

As a debugging interface SWD When you select JTAG Compare with table3.4 As shown in the figure, the number of terminals to connect the debugger is reduced, which has the advantage of reducing the connector mounting area on the board.

JTAGConnection terminal name	SWDConnection terminal name
TCK	SWCLK(TCK)
TMS	SWDIO(TMS)
T.D.O.	Not used
TDI	Not used
TRST	Not used

table3.4 Debug interface terminal name

However, all 2 Two-way communication JTAG Although it is not completely equivalent to Core Sight It can correspond to Debugger manufacturers provide optional connectors that are compatible with both, so you will need to select the interface that your device has. TMMP332FWUG teethSWD Built-in debug interface.

3.3.3Trace function

The trace function ETM(Embedded Trace Macrocell) and SWV(Serial Wire Viewer)of2 There are different types.ETM Regarding tracing CPUtoETM cannot be used without this unit.

Both allow tracing without affecting the CPU program, but there are differences in the timing and content of the trace. TMMP332FWUG supports ETM and SWV tracing.
Table 3.5 ETM trace and SWV trace overview.

	ETM	SWV
Required wiring	2～5 pieces	22 pieces
Impact on user programs	none	none
trace timing	Program counter transition timing	clock frequency
Trace accuracy	The error is extremely small	Because it depends on the clock frequency, the error increases if the program counter has many transitions.
Timestamp accuracy	There is an error	No error
Missing trace data	Proportional to the number of wires	Large amount of data or depends on host PC performance

Table 3.5 ETM trace and SWV trace summary

Below are the characteristics of ETM traces and SWV traces.

3.3.3.1 About ETM trace
The timing diagram of ETM trace is shown below.

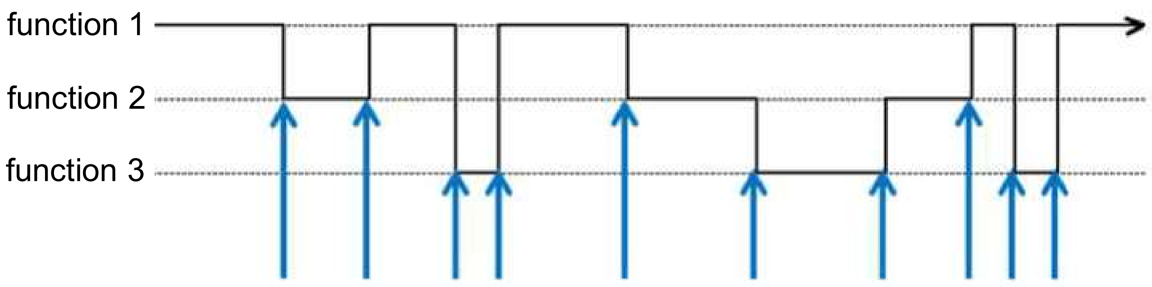


Figure 3.9 ETM trace timing diagram

If you look at the ETM trace timing diagram in Figure 3.9, you can see that the data trace timing is traced at the timing of a function transition (program counter change). Therefore, it is possible to understand the execution path of the program, and with high real -time performance, it is possible to trace the status of the system when it is operating at maximum speed.

3.3.3.2 About SWV tracing
A timing diagram of SWV trace is shown.

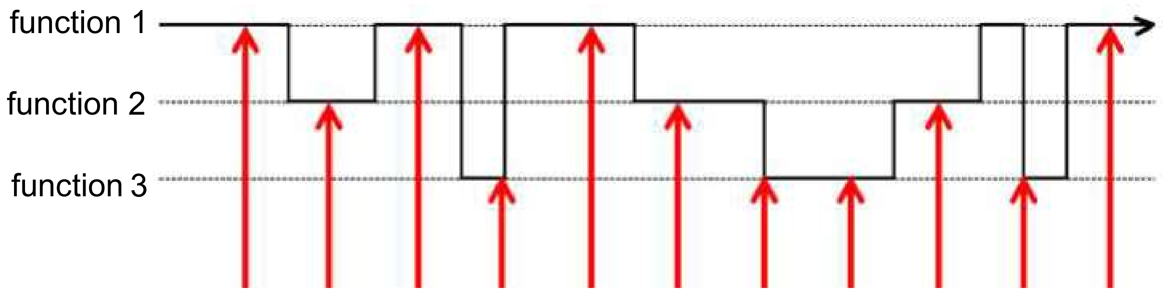


Figure 3.10 SWV trace timing diagram

If you look at the Figure 3.10 SWV trace timing diagram, you will notice that the trace spacing is always the same. This is because the SWCLK signal is used to set the trace spacing. Therefore, it can be said that tracing is not possible for functions that finish faster than the SWCLK speed. As a countermeasure, it may be possible to deal with this by increasing the SWCLK clock speed, but this will increase the amount of data and may cause trace data to be lost.

3.3.4 Limitations of real-time debugging

The real-time debugger has useful functions such as breakpoints and step operation. However, when debugging embedded devices, the CPU operation and I/O operation do not match, so even if you step monitor the CPU side, you cannot stop the I/O. Since the main task of embedded devices is I/O control, this debugging work also requires some technique. For example, you can store the I/O operation history and CPU operation status in memory, and perform a memory dump of one cycle of I/O operations.

Chapter 4 Program structure

This chapter explains the program structure for embedded control equipment.
PC programs running on Windows OS have large task processing units, so they often keep the operator waiting while the task is being executed.
However, with embedded programs, the object to be controlled is a machine, and programs are written to ensure the execution speed required by the machine, without making the machine wait. Although it is possible to write a control program using an embedded OS such as μITRON, here we will explain the structure of a program that independently controls a microcomputer.

4.1 Task control

Let's use examples of single-tasking and multi-tasking programs to understand the problems with single-tasking structures.

4.1.1 Traffic light control using single task

signal push button

roadway signal

Pedestrian Signs

Photo 4.1 Traffic light

siren

patrol light

Security push button SW

Photo 4.2 Security siren・pat light

Photo 4.1 shows a push-button pedestrian signal, and the roadside signal is blinking yellow.

Photo 4.2 shows a security light warning device installed on a traffic light pole that is only used in certain areas of Japan.

If you press the push button SW attached to the traffic light pole when you are being followed by a suspicious person, the siren and patrol lights will intimidate and repel the suspicious person.

First, let's consider the traffic light program.

Figure 4.1 shows the single-task control flow diagram for the traffic light shown in Photo 4.1, which has a pedestrian push button (PB) switch, the road side is flashing yellow, and the signal is waiting for an input to the pedestrian push button (PB) switch.

Signal light display status (1) and (2) are blinking yellow. A 1 second loop timer is running in a loop to keep track of the time while monitoring the status of the pedestrian push button (PB) switch between blinking display states (1) and (2). In display status (3), (4), and (5), the signal changes from blue to yellow to red, but the loop timer is still taking time during this time.
Next, let's consider the problems with the single-task structure.
① It is necessary to write the same routine in multiple places, such as checking the pedestrian push button (PB) switch and loop timer, making maintenance time-consuming.
②The computer is only killing time with a loop timer and is not doing any real work.
③ When writing additional programs, the same routine must be added to each loop timer. When a program is added to the loop timer, the loop timer value changes, and the counter value must be adjusted each time.
Here, we used the waste of a loop timer as an example to explain why programs with a single-task structure are not practical. Did you understand?
Next, let's add a security program to the flow diagram in Figure 4.1.
Let's consider a program that constantly monitors the state of the security push button SW and activates the siren and patrol lights for 3 minutes when the security push button SW is pressed.
It will be difficult!

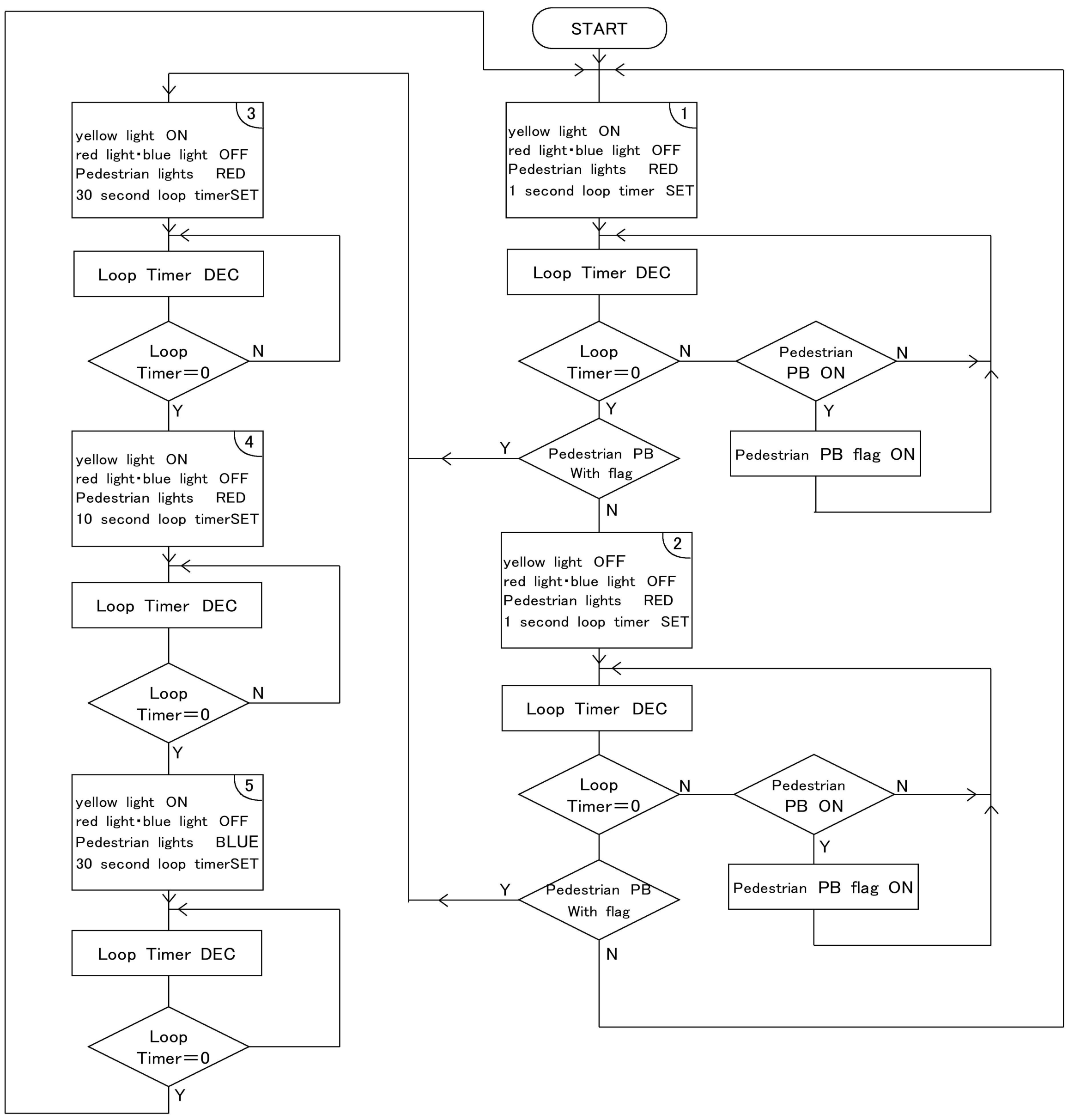


Figure 4.1 Single-task signal control flow diagram

4.1.2 Traffic light control by multitasking

Multitasking without an OS creates an infinite loop in the main routine as shown in Figure 4.2, and while going around in the main routine, it checks the start conditions of each task (thread), and if the conditions are met, it executes the corresponding task (thread). In assembly code, an infinite loop is created by returning from the last line of the main routine to the first line with JP MAIN. When written in C language, for(;;) or while(1) is used within main( ).

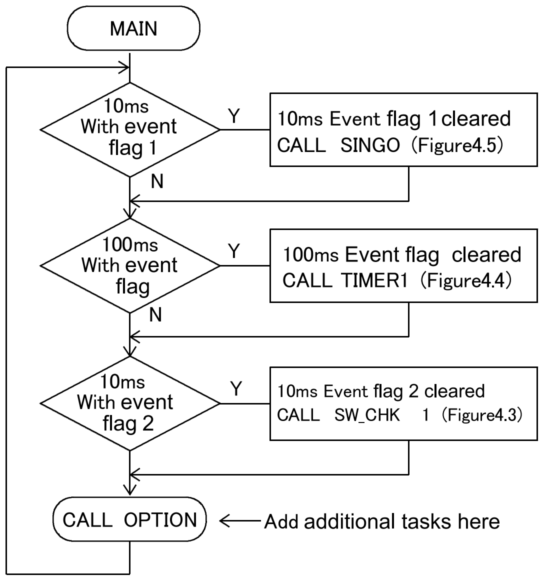


Figure 4.2 Signal control main routine

In the main routine for signal control shown in Figure 4.2, the 10ms, 100ms, and 10ms time event flags are checked in sequence while looping around in the main routine. When a time flag is set , the program goes to the corresponding thread, first turns the time flag OFF, executes the thread once, then returns to the main routine and resumes checking the time flag. In this way, each task (thread) SINGO, TIMER, and SW_CHK is executed. You can also add any program to the CALLOPTION position in Figure 4.2. Section 4.1.3 shows an example of adding a crime prevention program (BOUHAN).

Next, we will explain the SW_CHK, TIMER, and SINGO threads.

(1) SW_CHK1

It checks the status of the pedestrian push button (PB) switch, and if it is pressed, it turns the pedestrian PB flag 'ON' (Figure 4.3).

(2) TIMER1

The timer counter value is decremented in 0.1 second increments until it reaches 0 (Figure 4.4).

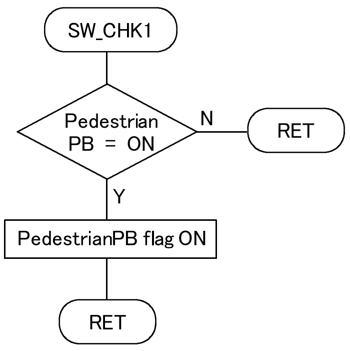
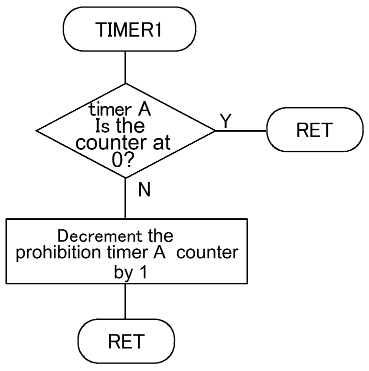


Figure 4.3 Pedestrian PB check



TIMER1
timer A Is the counter at 0?
Y
RET
N
Decrement the prohibition timer A counter by 1
RET

Figure 4.4 Timer A Counter DEC

(3) SINGO (Fig.4.5)

This controls the lighting status of the signal. The display status of the signal is managed by assigning a status number as shown in Table 4.1. Please refer to the display number in Figure 4.1.

No.	Traffic light status
0	Before display settings
1	Yellow flashing traffic light yellow light 'ON'
2	Yellow flashing traffic light yellow light 'OFF'
3	Green light 'ON' on a green-yellow-red traffic light
4	Yellow light 'ON' on a green-yellow-red traffic light
5	Green, yellow and red traffic lights in red 'ON' state

Table 4.1 Traffic light status

After waiting for the time set by the execution prohibition timer, the traffic light will perform the following actions:

①Normally, the roadway signal is flashing yellow, and the pedestrian signal is red (no crossing), and it is waiting for an event in which the pedestrian's push button (PB) switch is pressed. State 1 ⇔ 2②When the pedestrian's push button (PB) switch is pressed, the pedestrian PB flag turns 'ON', control moves from state 1, 2 to state 3 , 4, 5, and state 3 is set. ③From each state of state 3, 4, 5, the execution prohibition timer waits for a time, the roadway signal switches from green to yellow to red, and when the roadway signal is red, the pedestrian signal turns green, then returns to flashing yellow, and state 1 ⇔ 2 is repeated, waiting for a push button (PB) input.

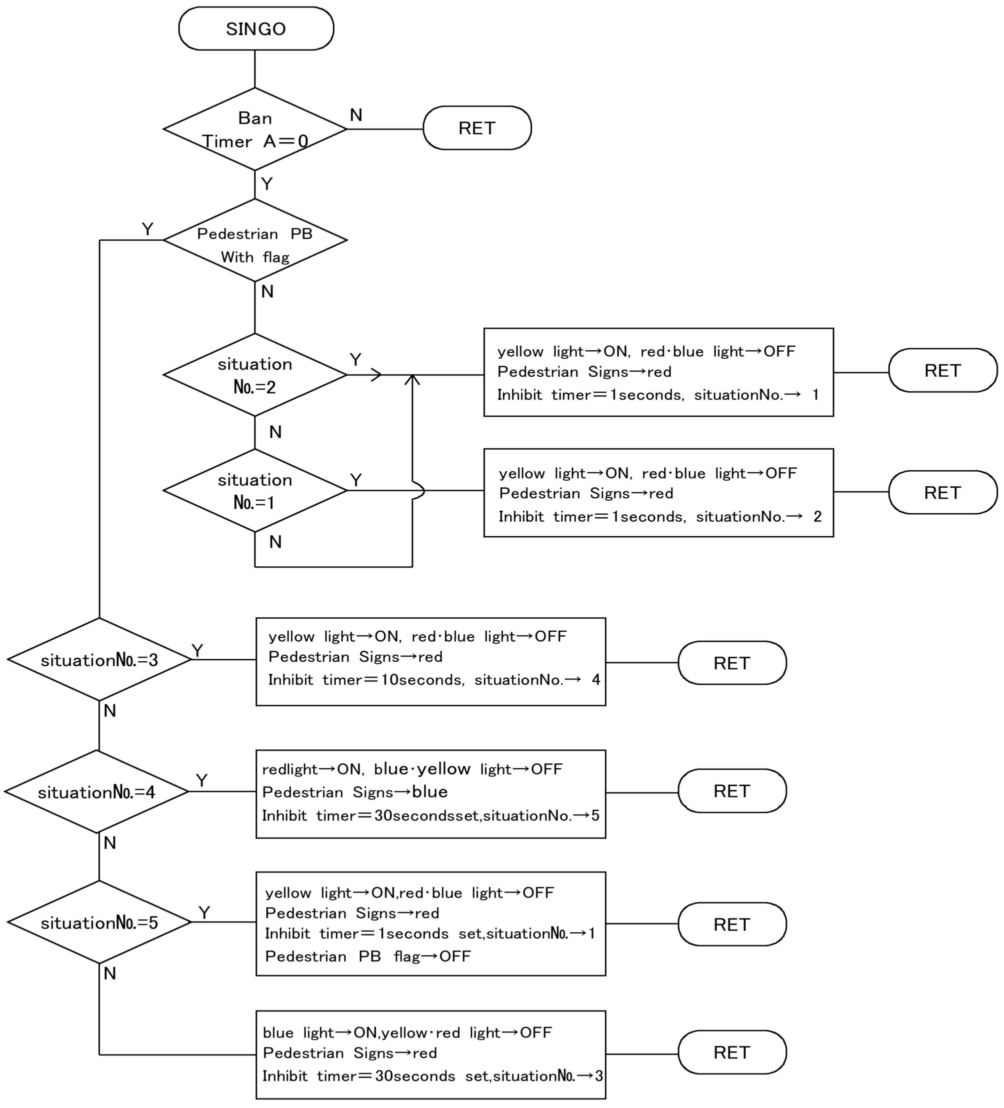

SINGO
Ban Timer A=0
N
RET
Y
Pedestrian PB With flag
Y
N
situation No.=2
Y
yellow light→ON, red·blue light→OFF
Pedestrian Signs→red
Inhibit timer=1seconds, situationNo.→ 1
RET
N
situation No.=1
Y
yellow light→ON, red·blue light→OFF
Pedestrian Signs→red
Inhibit timer=1seconds, situationNo.→ 2
RET
N
situationNo.=3
Y
yellow light→ON, red·blue light→OFF
Pedestrian Signs→red
Inhibit timer=10seconds, situationNo.→ 4
RET
N
situationNo.=4
Y
redlight→ON, blue·yellow light→OFF
Pedestrian Signs→blue
Inhibit timer=30secondsset,situationNo.→5
RET
N
situationNo.=5
Y
yellow light→ON,red·blue light→OFF
Pedestrian Signs→red
Inhibit timer=1seconds set,situationNo.→1
Pedestrian PB flag→OFF
RET
N
blue light→ON,yellow·red light→OFF
Pedestrian Signs→red
Inhibit timer=30seconds set,situationNo.→3
RET

Figure 4.5 SINGO control by state number

4.1.3 Adding a security program

The CALLOPTION section of Figure 4.2 shows an example of adding a program called BOUHAN that activates a siren and a police light for three minutes when the security push button is pressed.

Addendum 1: Following SW_CHK1, add Figure 4.6 Security PB Check SW_CHK2.
Addendum 2: Following TIMER1, add Figure 4.7 Timer B Counter DEC TIMER2.

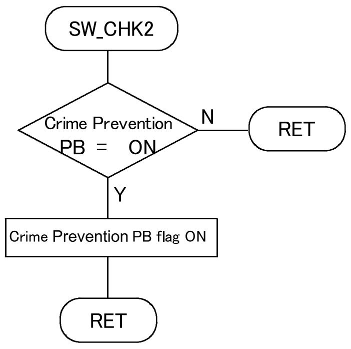


Figure 4.6 Crime Prevention PB Check

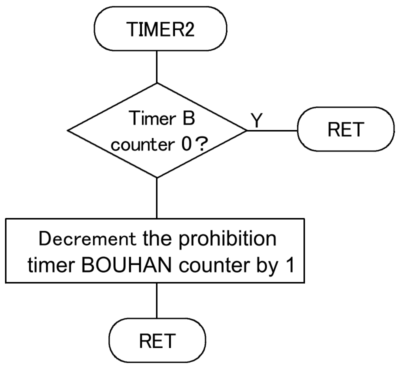


Figure 4.7 Timer B Counter DEC

Addendum 3: Add BOUHAN to the CALL OPTION position in the same way as SINGO, using a 10ms event flag. The crime prevention program also uses state numbers. State number 0 is the switch input waiting state, and state number 1 is the siren and police light are activated, and the crime prevention program is in progress.

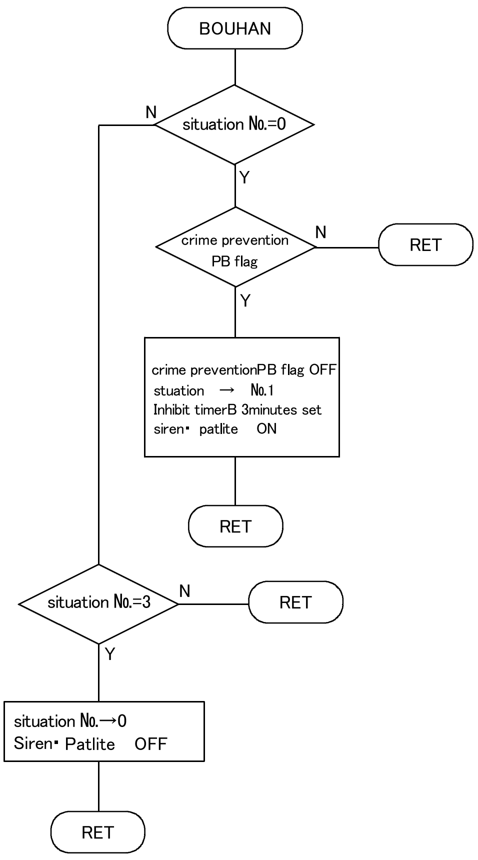


Figure 4.8 Crime prevention control program

★★★summary★★★

To maximize the performance of a microcomputer system, one task must not monopolize CPU resources. Each task and interrupt uses a single CPU at a set time (time sharing), so the golden rule is that each task must borrow the CPU from the main routine and immediately return it to the main routine when it has finished its work. The CPU must not be used as a loop timer or to wait for I/O. CPU resources are the shared property of the computer system.

4.2 Program Structure

When dealing with embedded programs, it is essential to understand the transition from CPU startup to main(), as well as the multitasking that you create yourself by manipulating the main routine, interrupts, and subroutines. Even when configuring multitasking using an embedded OS such as μ ITRON or Linux, the primitive multitasking introduced here is the starting point.

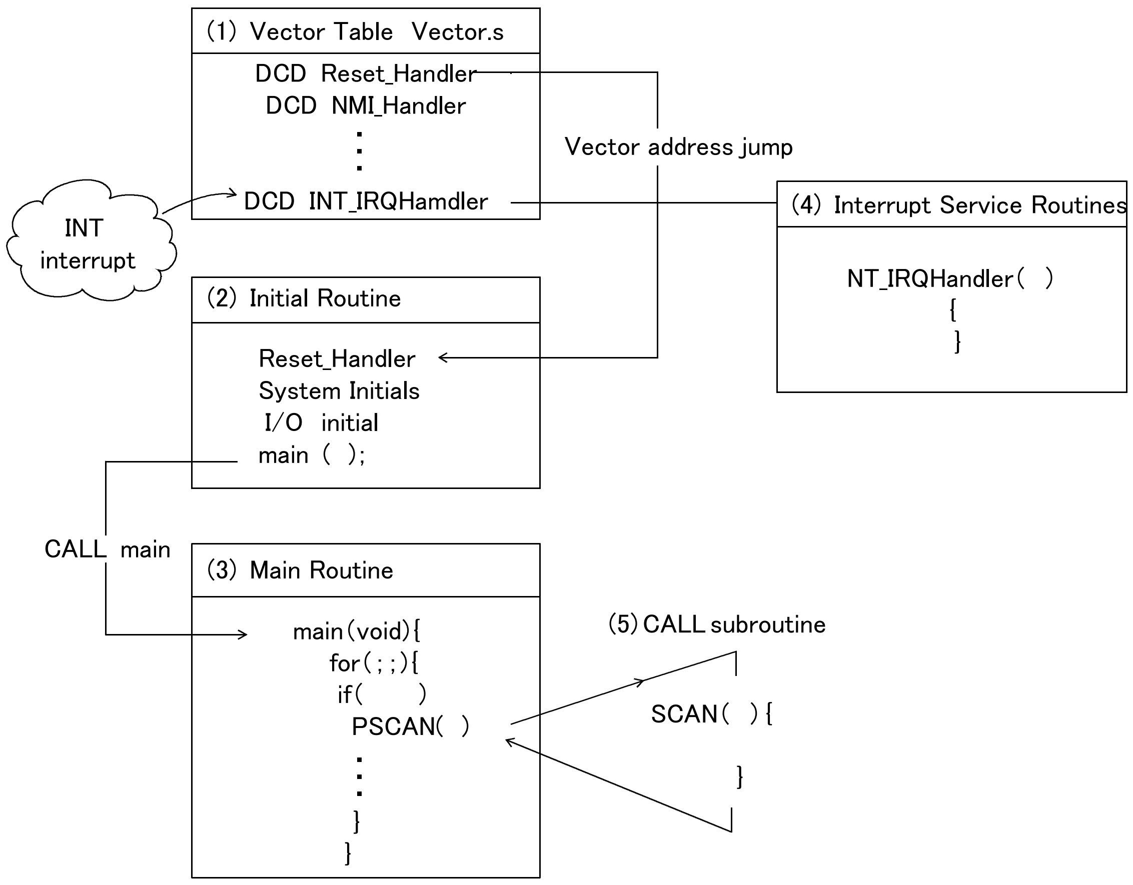


Figure 4.9 Program Structure (1)

4.2.1 Primitive multitasking

Since both the TLCS-900 and the Cortex-M3 are vector-activated, the program structure is composed of (1) the vector table to (5) subroutines as shown in Figure 4.9, and the transitions from startup to main( ) and within main( ) are also performed in the same way.

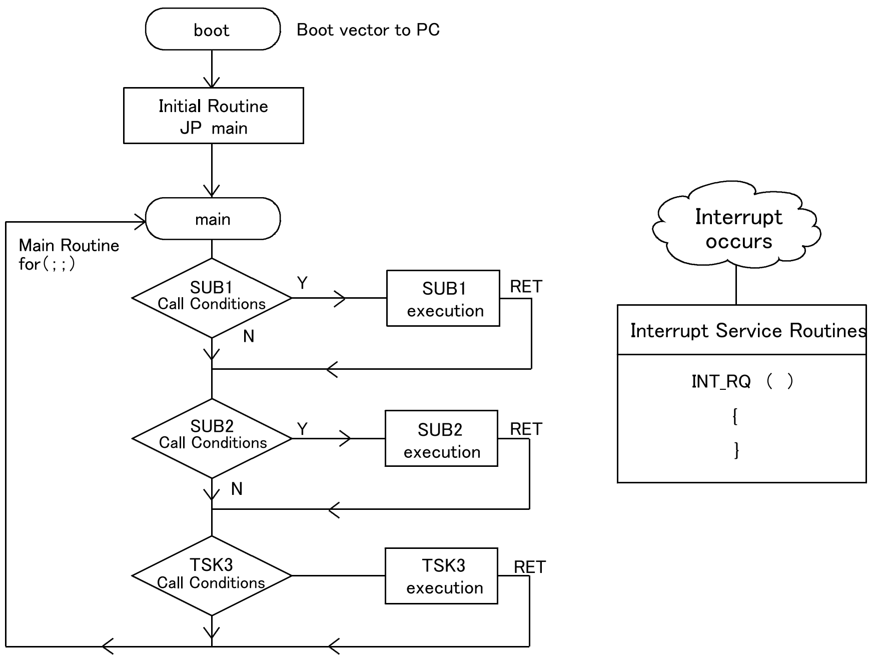


Figure 4.10 Program Structure (2)

Primitive multitasking enters the main routine after the CPU is started and initialized, as shown in Figure 4.10. The main routine is an infinite loop that goes around and around, checking the execution conditions of tasks, subroutines, etc., and executing them if the conditions are met.

Also, when an interrupt request occurs, the corresponding interrupt service routine is executed. Multitasking is achieved by distributing CPU resources smoothly among tasks, subroutines, and interrupts executed from the main routine, and using them skillfully. Next, we will explain each program block (1) to (5) in Figure 4.9. Please use an editor to visually trace the included program and read the overview of each program block.

(1) Vector table

The vector table is a list of jump destinations when an interrupt occurs. The TLCS-900 has the vector table located at address FFFF00H , while the Cortex-M3 has the vector table located at address 0 by default.

List 4.1 Example of a vector table for Cortex-M3

```
vector_table    DCD    STACK              ;Stack Pointer
                DCD    Reset_Handler      ;Start address
                DCD    NMI_Handler        ;Non-maskable interrupts
                        .
                        .
                        .
                DCD    INT_IRQHandler     ;interrupt
                        .
                        .
                        .
```

In the case of Cortex-M3, a vector table with one data item of 4 bytes is placed from address 0H as shown in List 4.1. Address 0H is the initial value of the stack, and Reset_Handler at address 4H describes the start address of the initial routine in which the system initials and I/O initials are described as shown in Figure 4.9. Copying this value to the program counter jumps to the start of the initial routine. Addresses up to 3CH are system interrupt frames such as debug interrupts, and from address 40H onwards, interrupt vectors from I/O provided by the device vendor are placed.

List 4.2 Vector table example for TLCS-900

```
FFFF00H    START       ;Startup Vector
FFFF04H    SWI_1       ;Software Interrupts1
FFFF08H    SWI_2       ;Software Interrupts2
FFFF0CH    SWI_3       ;Software Interrupts3
FFFF10H    SWI_4       ;Software Interrupts4
FFFF14H    SWI_5       ;Software Interrupts5
FFFF18H    SWI_6       ;Software Interrupts6
FFFF1CH    SWI_7       ;Software Interrupts7
FFFF20H    NMI_INT     ;Non-maskable interrupts
FFFF24H    WDT_TIME    ;Watchdog Timeout
FFFF28H    INT0_Z      ;INT0External Interrupts
FFFF2CH    INT1_Z      ;INT1External Interrupts
```

The TLCS-900 vector table is located from address FFFF00H as shown in List 4.2, with the system-related vector area up to address FFFF1CH and the normal interrupt vector area from there.

(2) Initial routine

The actual allocation address value of the first line of the initial is written in the reset vector, and when the CPU is reset, the reset vector is copied to the program counter and the fetch of the initial program begins. The initial setting of the embedded CPU first sets the multiplication of the system clock oscillation circuit and operates the CPU with the normal clock. This is a method of creating a high-frequency system clock by analog processing of a clock with an original oscillation of about 10MHz. For example, multiplication such as multiplying the original oscillation of 10MHz by 8 to create 80MHz, and then dividing it by 2 to create a 40 MHz system clock to adjust the waveform is commonly done in embedded CPUs. Next, set the dual-purpose terminal to a dedicated I/O terminal or a general - purpose input/output port according to the purpose, and then perform detailed I/O settings. This work should be done carefully according to the purpose of use, with the instruction manual of each manufacturer in hand. Finally, jump to main( ).

(3) Main routine
As already explained in Figure 4.9 and Figure 4.10 "Program structure", the main routine goes round and round while waiting for the condition to be called by the subroutine.
This round and round seems wasteful at first glance, but it is the best way to divide the CPU resources equally among each routine and task.
When each routine or task finishes its work, it promptly returns the right to use the CPU to the main so that other routines and tasks can use the CPU immediately
.
By not monopolizing the CPU, the entire program can run smoothly.
(4) Interrupt service routine
When an interrupt request occurs, the main routine in Figure 4.9(3) stops the work it is running round and round, and the interrupt service routine in Figure 4.9(4) described in Figure 4.9(1) Interrupt table address specified by interrupt cause is executed as a priority.
The interrupt execution sequence will be described later. Interrupt service routines are extremely nuisances because they forcibly take away the right to use the CPU from the main routine without any delay.

Write the program in the interrupt service routine simply and return the right to use the CPU to the main routine as soon as possible.

(5)Interrupt execution sequence

An interrupt is a subroutine call whose jump destination is specified in advance by a vector table, etc.

When executing an interrupt, various procedures such as register evacuation are required, and generally, the TLCS-900 is determined by the programmer, while the Cortex-M3 is determined by the compiler, as follows.

-TLCS-900 interrupt execution sequence

1)Interrupt cause occurs

2)Read interrupt vector value

3)PUSH PC, PUSH SR, interrupt level +1 to current

4)PC ←vector value

5)Interrupt processing program starts from here

Evacuate registers as necessary

Evacuate registers at the discretion of the programmer using the PUSH command or switching register banks

Execute interrupt processing

POP for PUSH of register evacuation is performed here.

6) RETI command execution
POP SR, POP PC, register bank restoration, interrupt level restoration is automatically performed before interrupt
7) Interrupt sequence ends

・Cortex-M3 interrupt execution sequence

1) Interrupt factor occurs

2) Stacking

When an interrupt is detected, the compiler automatically stacks in the order PC, PSR, R0, R1, R2, R3, R12, and LR.

This allows the interrupt handler to be written as a normal C function.

3) Vector fetch

Since the Cortex-M3 is a Harvard architecture, stacking (using the data bus) and vector fetch (using the instruction bus) are performed at the same time.

4) Preparation and execution of interrupt processing

SP and PSR are updated according to interrupt execution

Vector address is copied to PC

LR is updated for interrupt

Interrupt processing is executed

5) Return from interrupt

When the compiler detects the last } of a C function, it automatically issues BX LR: POP [xx], etc., and unstacking, etc. is also executed.

4.2.2 Task Management

In the primitive multitasking structure shown in Figure 4.10, the programmer manages multiple subroutines and tasks using the following methods:

(1) Inter-task access

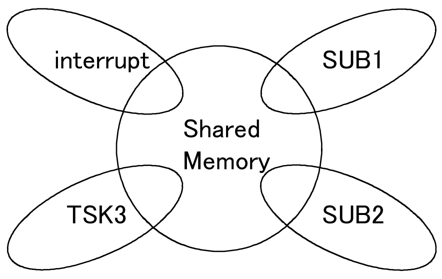


Figure 4.11 Inter-task access

Each task is linked by a global shared memory, as shown in Figure 4.11.

You need to be careful about when you rewrite and reference the shared memory.

There is a big difference in speed between tasks managed by I/O and tasks managed by the CPU, so use an I/O ready flag (similar to a semaphore) to share data.

(2) Priority

The priority between tasks in a primitive multitask is managed by the interrupt request level and interlocks within the main routine, as shown in Figure 4.12.

As shown in Figure 4.13, if an interrupt occurs while a task in the main routine is being executed, the task being executed in the main routine is interrupted and the interrupt task is forcibly executed. Furthermore, if an interrupt with a higher priority occurs, the currently executing interrupt task is also interrupted, and the higher priority interrupt task is executed first.

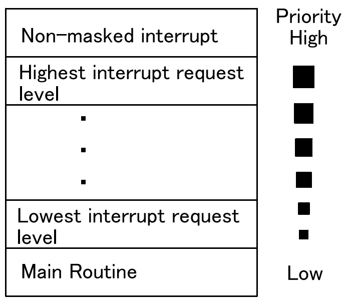

Non-masked interrupt
Highest interrupt request level
Lowest interrupt request level
Main Routine
Priority High
Low

Figure 4.12 Execution Precedence

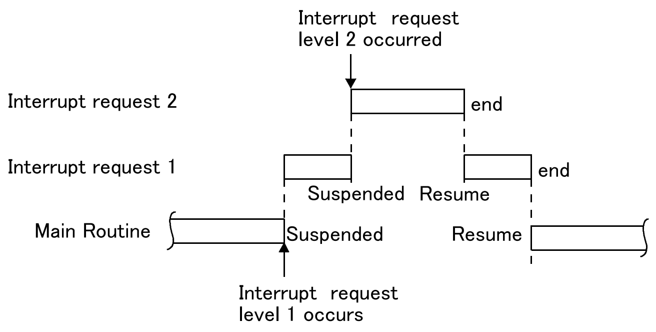


Figure 4.13. Overriding Interrupts

4.3 Assembler and C Compiler Behavior

The TLCS-900 version of this bicycle teaching material writes the same bicycle control program in assembly language and C language, and the label names of both languages are unified as much as possible, so it is easy to compare the two languages. Furthermore, you can read the assembly file generated by C language, so by comparing a program written in assembly language with the assembly generated by C, you can see the structure of C language.

This is a very interesting teaching material for understanding program structure.

By reading this section, you can understand how to write arguments and return values of C functions and how to use stack frames.

4.3.1 Arguments and return values (1)

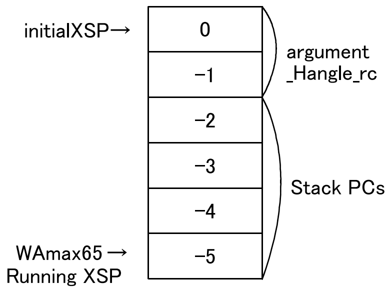
Specific examples of function arguments and return values are explained in /*Check manual steering angle deviation*/ HND_HDL(void) in List 4.3, List 4.4, and List 4. 5.

List 4.3 is an example of a function call written in C language. The function WA_ max65 (int wa) is called with an argument (Hangle-re) and an int return value is obtained.

List 4.3 Arguments and return values (1) C language source

```
/*Manual handle angle deviation check*/
    void  HND_HDL(void){ int      wa;              //  Manual Handle Control
                                                   //

          Hangle_buf  =  WA_max65(Hangle_rc);      //  Handle angle upper and lower limit adjustment
          wa  =  Hangle_buf  -  Hndl_feedback;     //  Deviation = Order - Feedback
          if  (wa  >=  0){                         //  Is the deviation positive?
          L_side(wa);                              //  Check deviation on left rotation side

          else{          }                         //  Is the deviation negative?
          R_side(wa);                              //  Check deviation on right rotation side
              }
                 }


/*Handle angle upper and lower limit adjustment*/
int WA_max65(int wa){ // Handle angle upper and lower limit adjustment
          if  (wa  >=  0){                         //  Right?
                 if  (wa  <  650){                 // Less than 650?
```

```
                return(wa);                          //Return it as is
                                  }
                return(650);                         //  Return 650
                       }                             //  RET
        else{                                        //  negative?
                if (wa >= -650){                     //  -650 or more?
                return(wa);                          //Return it as is
                                  }                  //  Return -650
                return(-650);                        //  RET
         }
               }
```



Listing 4.4 is the assembly listing generated by C.
The argument _Hangle_rc is passed via the stack and referenced by ld HL, (XSP+0x4).
The return value is returned in the HL register, so when returning from the function call, a stack frame adjustment inc 0x2,XSP is performed.

Figure 4.14 WAmax65 running stack

Listing 4.4 Arguments and return values (1) C compiler generated ASM

```
_HND_HDL:
                pushw   (_Hangle_rc)                    ;Arguments to stack push
                cal     _WA_max65                       ;Call Function
                inc     0x2,XSP                         ;Adjust stack frame by 2 bytes
                ld      (_Hangle_buf),HL
                sub      HL,(_Hndl_feedback)
                j       lt,L74
                push     HL
                cal     _L_side
                inc     0x2,XSP
                ret
L74:            push    HL
                cal     _R_side
                inc     0x2,XSP
                ret

_WA_max65:
                ld      HL,(XSP+0x4)                    ;See argument _Hangle_rc
                cp       HL,0x0
                j       lt,L47                           ;650
                cp       HL,0x28a
                ret     lt
                ld      HL,0x28a                        ;650
                ret
L47:            cp      HL,0xfd76                       ;64886/-650
                j       lt,L49
                ret                                      ;64886/-650
L49:            ld      HL,0xfd76
                ret
```

List 4.5 is an example of direct address-specified assembly language that contains the same content as the C language source in List 4.3.
When writing programs in assembly language, arguments and return values are generally passed in registers. Variables that are specified with a register width type at the start of a block of tasks are reused as much as possible within that task (as local variables) and the final results are stored in memory (as global variables). Although there are portability issues, this generates the most efficient code.

List 4.5 Arguments and return values (1) Assembly source

```
HND_HDL:  LD      BC,(1070H)          ;LD  Handle Feedback+-1023
          LD      WA,(1034H)          ;LD  Handle Order  +-650
          CAL     WAmax65             ;MAX  CHK, +-  700  -> 800 The following
          LD      (1036H),WA          ;to  Handle Order Output  BFF
          SUB     WA,BC               ;Comment = Order - Feedback
          J       GT,L_side           ;Left hand side of the wallCHK
          J       R_side              ;Right hand side of the  CHK
          RET


          ; MAX  CHK, WAｦ +- 650 ｲｶﾄ ｽﾙ
WAmax65:  CP      WA,650              ;Plus River  max  CHK
          J       GT,P650WA           ;max
          CP      WA,-650             ;Minus Rivermin  CHK
          J       LT,N650WA           ;min
          RET
P650WA:   LD      WA,650
          RET
N650WA:   LD      WA,-650
          RET
          ;
```

4.3.2 Arguments and Return Values (2)

The handling of arguments and variables will be explained using the 10m x n countdown timers TIM10 and AUX_T.

List 4.6 shows the 10m x n countdown timer TIM10( ), the countdown portion of the timer counter AUX_T( ) called by TIM10( ), and an example of how to use the TIM10( ) timer RtrnLIM( ).

TIM10( ) is a task that starts once every 10ms.

From here, call the down counter AUX_T(T_10ms,4) with the 4-byte array T_10ms as an argument.

RtrnLIM( ) is set to 10ms x 50 = 500ms, and write a program that runs once every 500 ms.

List 4.6 C language source for 10msec x n timer

```
/*Right steering angle limit correction addition*/  // Example of using a 10msec x timer
void RtrnLIM(void){                    // Add right turn steering angle limit
    if (T_10ms[1] != 0){               // Steering angle limit prohibition timer is not 0?

        return;                        // escape
    }

    T_10ms[1] = 50;                    // Steering angle limit prohibition timer=50(500ms)
                                       //Write a program that runs once every 500ms here
}
------------------------------------------------------------------------------------------
/*10msecTimer Processing*/
void TIM10(void){                      // 10msTimer Processing
    T10ms_flag &= 0xfb;                // 10msTimer Flag Reset

    AUX_T(T_10ms, 4);                  // Timer subtraction (set 0 to 4 bytes from T_10ms)
}
------------------------------------------------------------------------------------------
/*Timer value batch setting*/
/*Various timers are counting.*/
void AUX_T(unsigned char *t_cnt, unsigned char c){

    while (c > 0){
        if (t_cnt[c-1] != 0){
            t_cnt[c-1]--;
        }
        c--;
    }
}
```

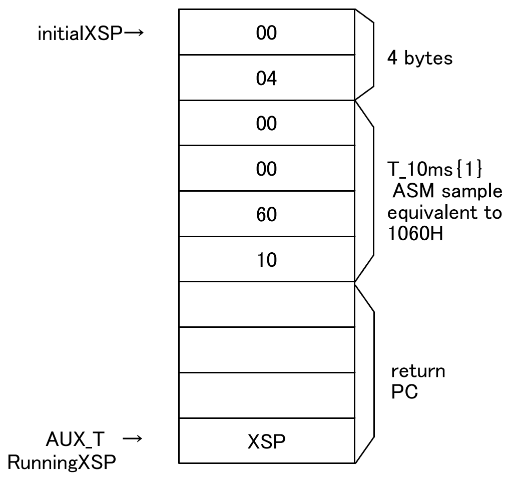


Figure 4.15. Stack during AUX_T execution

List 4.7 is the assembly listing generated by the C language source in List 4.6.

The 6-byte argument is pushed three times, passed to AUX_T via the stack, and referenced by (XSP+0x8) and (XSP+0x4).

There is no return value, and the globally declared array T_10ms{4} is directly rewritten, so stack frame adjustment inc 0 x6,XSP is performed when returning from the function.

The TLCS-900 is a CPU separated into a register machine, and addressing via registers and directly to memory is powerful, but stack addressing is still a bit weak. In my personal opinion, I don't see much benefit in the TLCS-900 compiler passing arguments via the stack.

Listing 4.7 C compiler generated ASM for a 10ms x n timer

```
_RtrnLIM:                                   ;1Example of using 0msec x n timer
      lda       XBC,_T_10ms + 0x1           ;Check the current location of the timer counter [1]
      ld        A,(XBC)
      cp        A,0x0
      ret       ne                          ;If not 0, RET

L79:  ;1
      ldb       (XBC),0x32                  ;Timer counter[1]←Reset 50
      ret                                   ;Write a program that runs once every 500ms here
-----------------------------------------------------------------------------------------
_TIM10:
      res       0x2,(_T10ms_flag)           ;Called with the 10 ms flag and turns the 10 ms flag OFF
      pushw     0x4                         ;Pass 4-byte array number as argument
      pushw     _T_10ms >> 16               ;Pass the top of the array as an argument twice
      pushw     _T_10ms & 0xffff
      cal       _AUX_T                      ;Call timer subtraction routine
      inc       0x6,XSP                     ;Stack frame adjustment 6 bytes

      ret
-----------------------------------------------------------------------------------------
AUX_T:
      ld        E,(XSP+0x8)                 ;Sequence number reference


      cp        E,0x0
      ret       eq
L4:   ;2
      ld        A,E
      extz      WA
      dec       0x1,WA                      ;Number of sequences-1
      ld        XBC,(XSP+0x4)               ;(Number of sequences - 1) + sequence top
      lda       XBC,XBC+WA                  ;XBC=Array pointer
      ld        A,(XBC)                     ;DEC each array to 0
      cp        A,0x0
      j         eq,L5
      dec       0x1,A
      ld        (XBC),A
L5:   ;2
      dec       0x1,E                       ;Decimate the number of sequences
      j         ne,L4                       ;0 to RET
L3:   ;1
     ret
```

Listing 4.8 is an example of direct address specified assembly language that contains the same content as the C language source in Listing 4.6.
Comparing Listing 4.7 and Listing 4.8, you can see that using assembly language is overwhelmingly more efficient and easier to debug for compact embedded programs that do not make heavy use of C language library functions.
However, assembly language cannot handle areas that require OS dependency, such as Ethernet and ATA.
When developing embedded devices, choose the right CPU, development language, and OS for the right place. Bigger does not fit smaller, and smaller does not fit bigger.

Listing 4.8 Assembly source for a 10ms x n timer

```
;Right turn steering angle limit increment; Example of using 10msec x n timer

 RtrnLIM:   CP    (1061H),0          ;10msec counter check
            RET   NZ                 ;If not 0, RET
            LD    (1061H),50         ;Reset 10msec x 50
                                     ;Write a program that runs once every 500ms here
            RET
            ;
------------------------------------------------------------------------------------
;Called from the main routine with a 10msec trigger
TIM10:      RES   2,(1002H)          ;10m seconds flag OFF
            LD    XHL,1060H          ;Array top address
            LD    W,4                ;Number of sequences
            LD    A,0                ;DEC to 0
            CAL   AUX_T              ;Call subtraction routine

            RET
------------------------------------------------------------------------------------
AUX_T:      CP    A,(XHL)            ;Check the contents of an array
            J     Z,LVAUX            ;0 at JP LVAUX

            DEC   1,(XHL)            ;Not 0, 1DEC
LVAUX:      INC   1,XHL              ;Array pointers INC
            DEC   1,W                ;Number of sequences DEC
            J     NZ,AUX_T;If the number of arrays is not 0, JP AUX_T

            RET                      ;0 to RET
```

4.3.3 Type conversion by casting

Variables used in C language programs are declared in advance, but there are cases where a temporary type change is required due to program or compiler reasons.

The cast operator is used to change the type of a variable and perform a forced type conversion.

The following example of using a cast is shown using multiplication and division of INT type.

In assembly language, the concept of variable type and size is not very prominent, so multiplication and division are expressed using the following types and sizes, and the presence or absence of a sign is determined by the most significant bit of the instruction and data.

Signed multiplication MULS dst,src, unsigned multiplication MUL dst,src

Signed division DIVS dst,src, unsigned division DIV dst,src

Multiplication: 16 bits x 16 bits = 32 bits

Division: 32 bits ÷ 16 bits = upper 16 bits are the remainder, lower 16 bits are the quotient

Since the C compiler has some indeterminate elements when mixing variable sizes, the following Lists 4.9, 4.10, and 4.11 explain examples of how to deal with this using cast operators.

S3S4def( ) in List 4.9 uses the variable wbc declared as an int to perform multiplication and division as in the assembly in the comment line.

S3S4chg( ) in List 4.9 uses the cast operator to specify the multiplication result as long and the division result as int.

Let's check the results in List 4.10.

List 4.9 Cast usage example C language source

```
int   S3S4def(int bc){                    //No cast conversion
          int     wbc;
          wbc = bc;
          wbc *= 5000;                    // MULS   XBC,5000
          wbc /= 1000;                    // DIVS   XBC,1000
          Correct_add += wbc;             // ADD   (1084H),BC
          return(wbc);                    // RET
          }

int   S3S4chg(int bc){                    //With cast conversion
          int    bc_buf;
          long  wbc;
          bc_buf = bc;
          wbc = (long)bc_buf * 6000;      // MULS    XBC,6000
          wbc = wbc / 1000;               // DIVS    XBC,1000
          Correct_add += (int)wbc;        // ADD     (1084H),BC
          return((int)wbc);               // RET
          }

int   S3S4(int bc){                       //
          int    wbc;
          if ((AD_flag & 0x04) != 0){      // BIT   2,(102FH) 1=CHG, 0=Default
          wbc = S3S4chg(bc);              // J  NZ,S3S4chg
          return(wbc);                    // RET
          }
          wbc = S3S4def(bc);              // J   Z,S3S4def
          return(wbc);                    // RET
          }
```

Listing 4.10 is the assembly listing generated by the C language source in Listing 4 .9.
_S3S4def: multiplies the argument passed as an INT by 5000, and then bit-extends the upper 16 bits of the answer from the lower 16 bits, so the 32 bits divided by 16 bits performed on the next line is not the intended result.
_S3S4chg: uses the C compiler library for multiplication and division, but casts the result of multiplication to long and the result of division to int, so the intended value is obtained for the results at the end.

Listing 4.10 Example of using casts with C compiler generated ASM

```
 _S3S4def:
          ld      WA,(XSP+0x4)
          muls    XWA,0x1388          ;*5000
          exts    XWA                 ;bitExtension
          divs    XWA,0x3e8           ;XWA/1000=qwa  remainder,wa
          add     (_Correct_add),WA
          ld      HL,WA
          ret


_S3S4chg:
          ld      WA,(XSP+0x4)
          exts    XWA
          ld      XBC,0x1770          ;*6000
          cal     C9H_mulls           ;Multiplication Library  xwa*xbc=xhl
```

```
            ld      XWA,XHL
            ld      XBC,0x3e8           ;/1000
            cal     C9H_divls           ;Division Library   xwa/xbc=xhl
            ld      WA,HL
            add      (_Correct_add),WA
            ret

_S3S4:    omission     ret
```

Listing 4.11 is an assembly language listing similar to Listing 4.9 above.
Division instructions always carry the risk of overflow.
The programmer must either manage the range of numbers used or take other measures, such as monitoring the overflow flag.

Listing 4.11 Assembly source for a cast example

```
S3S4:      BIT     2,(102FH)        ;FLG  CHK:  1=CHG,  0=Default
           J        NZ,S3S4chg      ;conditions,  SET,  S3  ON
           BIT     2,(102FH)        ;FLG  CHK:  1=CHG,  0=Default
           J        Z,S3S4def       ;Default,  RESET,  S4  ON
           RET
S3S4def:  MULS   XBC,5000           ;bc*5000=xbc
           DIVS   XBC,1000          ;xbc/1000=qbc,bc
            ADD   (1084H),BC
           RET
S3S4chg:  MULS  XBC,6000            ;*6000
           DIVS   XBC,1000          ;/1000
           ADD   (1084H),BC         ;xbc/1000=qbc,bc
            RET
```

4.3.4 Explanation of arrays
List 4.12, List 4.13, and List 4.14 are used to explain examples of compiling arrays.
Arrays are variables that can handle multiple data of the same type together.
In this example, we use the array declared as follows in the header file ram27def.h:

EXTERN unsigned char RX0_RES_BUF[6]; // RX0 receive buffer

The exclusive OR value of arrays [1] to [3] is compared with the checksum value of array [4] to check for errors in serial communication from the remote control.

This example shows only the checksum calculation part.

The C language source in List 4.12 uses an array to calculate the checksum. List 4.13 is an assembly list generated by the C compiler.

There is a little waste in the addressing.
List 4.14 is a checksum calculation list written in assembly language. The addressing uses "register indirect post-increment".

List 4.12 C source code

```
// TrmCHK() Checksum calculation using arrays Excerpt  //
     a = RX0_RES_BUF[1];                  // CHK SUM START ADDRESS
     a ^= RX0_RES_BUF[2];                 // XOR     A,(XHL+)
     a ^= RX0_RES_BUF[3];                 // XOR     A,(XHL+)
     if (a != RX0_RES_BUF[4]){            // Checksum abnormal? CP A,(XHL)
     M0SETR0();                           // Err Processing status reset
     return;                              // escape
```

Listing 4.13 C-Compiler Generated ASM

```
_TrmCHK: ;  Checksum calculation part excerpt
        lda     XIX,_RX0_RES_BUF                ;Start address of array RX0_RES_BUF
        ld      C,(XIX+0x1)                     ;CHK SUM START ADDRESS
        ld      W,C
        lda     XDE,XIX+0x2                     ;a ^= RX0_RES_BUF[2]
        ld      A,(XDE)
        xor     W,A
        lda     XHL,XIX+0x3                     ;a ^= RX0_RES_BUF[3]
        ld      A,(XHL)
        xor     W,A
        cp      W,(XIX+0x4)                     ;a != RX0_RES_BUF[4]
        j       ne,_M0SETR0
```

Listing 4.14 Assembly source

```
TrmCHK:  ;  Checksum calculation part excerpt
      LD        XHL,1301H                     ;CHK SUM START ADDRESS
      LD        A,(XHL+)                      ;1301H START CHR
      XOR       A,(XHL+)                      ;1302H
      XOR       A,(XHL+)                      ;1303H
      CP        A,(XHL)                       ;CHK CHKSUM
      J         NZ,M0SETR0                    ;Err
```

Chapter 5 Bicycle Control Architecture

This bicycle teaching material provides three types of programs that perform the same control: 1) C language program for ARM, 2) C language program for TLCS-900, 3) Assembly language program for TLCS -900. The function names and label names for each program are the same , and the control architecture is also the same, so here we will explain each program together, listing the related function names and label names for bicycle control.

5.1 Analog value initialization

The tilt and turning sensors used in bicycle teaching materials are relative value sensors that have a large temperature drift, so before riding the bicycle, the drift value must be corrected and the zero points of tilt and turning must be established with the bicycle standing upright and stationary. Specifically, ① after turning the bicycle's power on, ② set the bicycle on the starting platform so that the body is as upright as possible, and ③ press the stop button on the remote control, ④ the sensor's automatic drift correction and zero adjustment will be performed. During ④ the automatic correction, the LED lamp mounted on the bicycle's control board will flash, and when the automatic correction is completed, the LED lamp will go out and standby is complete. Next , ⑤ press the start button, and the LED will light up continuously, and from there ⑥ raise the accelerator knob and the bicycle will start moving.

5.1.1 Drift correction hardware

The tilt sensor and turning sensor used in the bicycle teaching materials are not absolute value sensors , but angular velocity sensors that output relative values. In addition, this angular velocity sensor is more susceptible to output changes due to temperature drift than to changes in sensor output due to changes in angular velocity, so as introduced in sections 2.1.1 and 2.1.2, it is a sensor that requires some ingenuity in how it is handled. This angular velocity sensor operates on a power supply voltage of 3 [V] and the output signal has a drift element of about ± 0.75 [V] centered on 1.35 [V], but the amplitude of the signal due to angular velocity is small and an amplification factor of 100 times or more is required.

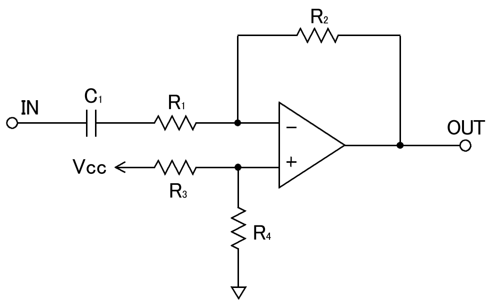


Figure 5.1 Op-amp AC amplifier circuit

As shown in Figure 5.1, it is possible to address this issue using an AC amplifier with an added coupling capacitor C1, but adding C1 will result in a high -pass filter (low-cut filter) that will sacrifice the frequency characteristics of the amplifier circuit. Therefore, this teaching material uses a method to correct the sensor drift using the output of a serial DAC controlled by the CPU in a DC amplifier, as shown in Figure 5.2.

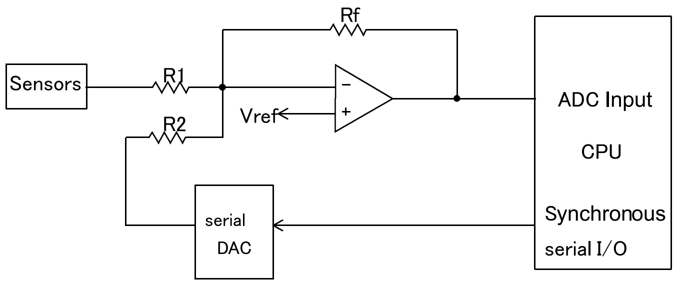


Figure 5.2 Schematic diagram of automatic drift correction circuit

Using an adder circuit consisting of an op amp , Rf, R1, and R2 in Figure 5.2, a polarity calculation is performed on the sensor output and DAC output with Vref as the reference, and the output of the serial DAC is adjusted so that the average input value of the ADC input terminal is near the center of the ADC input range (1 .65V).

5.1.2 Drift correction program

When a stop button switch command is received from the remote control, drift correction is performed in the following order. The drift correction sets the ADC input terminal voltage to about 1.65 [V], half of 3.3 [V], and the AD conversion value Turn_base, Slope_base or (1042H), (1044H) at that time becomes the reference value for the AD conversion input. After drift correction, the sensor input value - reference value becomes the analog input value with polarity. ① Check the remote control status bit RC_Sbit( ) Checks the stop order bit from the remote control and jumps to RC_STP( ) if there is an order ② Jumps from RC_STP( ) to Ana_RST( ), which prepares the analog reset flag, etc., and turns ON the reset flag for the tilt sensor and rotation sensor. The LED lamp starts flickering. ③ The 100ms flag routine TIM100() calls FLICK() to execute flicker and A_Reset() to execute analog reset at 100ms intervals. ④ FLCK() causes the LED lamp to flicker at 0.3s intervals while the flicker condition is met. ⑤ A_Reset() is called periodically from the 100ms routine and executes the following. If the tilt sensor reset flag is set, it calls KEI_RST() to reset the tilt sensor in step ⑥. If the rotation sensor reset flag is set, it calls SEN_RST() to reset the rotation sensor in step ⑦. If there is no tilt/rotation reset flag, the current steering wheel position is set as the steering wheel reference position and analog reset is complete. ⑥ KEI_RST() To find the tilt sensor correction value, it adjusts the serial DAC output value with KEI_INC() or KEI_DEC() to bring the ADC input value to around 1.65 [V], and when it is within that range, it turns the tilt sensor reset flag 'OFF'. The calculation of ADC input value - reference value = polarized slope value is performed within AD_AVE() to create a value of approximately ±500 bits, and this is then accumulated eight times to create a value of ±4000 bits.The ADC uses a 10-bit ADC built into the CPU, and converts digital output values from 0 to 102 3 bits into analog-to-digital conversion.

The average calculation routine AD_AVE() performs eight integrations to obtain a value between 0 and 8184 bits, and the reference value is subtracted from this value to obtain polarized data of approximately ±4000 bits. ⑦ SEN_RST() To find the correction value for the rotation sensor, the serial DAC output value is adjusted with SEN_INC() or SEN_DEC, the ADC input value is set to approximately 1.65 [V], and the rotation sensor reset flag is set to 'OFF' when it enters the range. The calculation of the polarized rotation value (ADC input value - reference value = polarized rotation value) is performed with TuenPID(), and the value is approximately ±4000 bits.

5.1.3 Notes on drift correction

The drift amount of the angular velocity sensor used in this bicycle teaching material can be large when the power is turned on or when the ambient temperature changes. As explained in section 5.1.2, drift correction is performed once when the stop button on the remote control is pressed in manual driving mode, so if the drift amount is extremely large, it will go out of the range of ±1.65 [V] in a short time and the bicycle will not be able to run normally. The bicycle will not stop turning despite the control from the remote control. If this happens , please stand the bicycle upright again and perform drift correction. The drift amount will stabilize once the temperature of the sensor element stabilizes.

5.2 Handle operation

The handlebar operation of this bicycle teaching material is configured with automatic balancing control that tracks the deviation between the order and feedback to zero, as shown in Figure 5.3. In addition, proportional control is also performed, which sets the tracking speed proportional to the deviation using a PWM motor driver. Here, we will list the names of functions related to automatic balancing using proportional control and explain mainly the software.

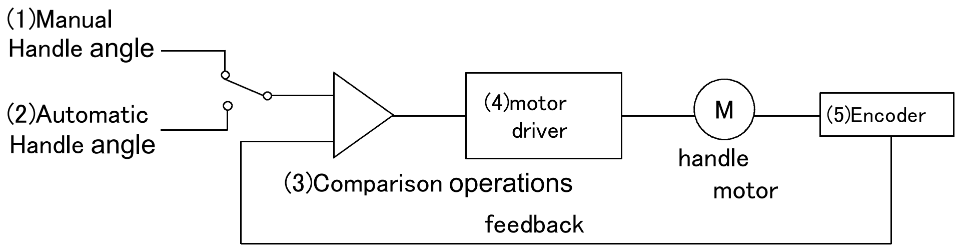


Figure 5.3 Handle operation block diagram

5.2.1 Overview of steering wheel control

We will now provide an overview of steering wheel control for the automobile teaching materials shown in Figure 5.

3. (1) Manual steering wheel angle order

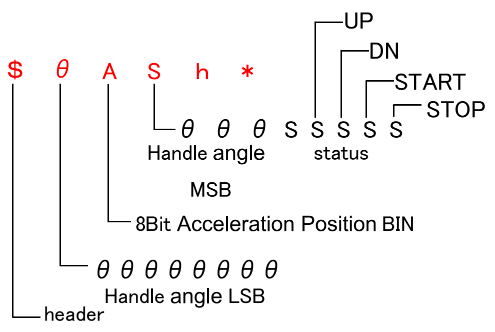


Figure 5.4 Remote control message

The remote control sends a six -character message starting with the header $ shown in Figure 5.4 via infrared communication, and θ in the message is the steering angle. This string is handled by the array RX0_RES_BUF[ ] or the (1200H) buffer. The string looks like this:

$ This is the header. 0x24

θ The handle angle

The 8 bits of the steering angle θ byte and the upper 3 bits of the status byte (a total of 11 bits) are mixed in HDLangl() and the ±800 -bit (1) manual steering angle is stored in Hndl_angle or (1034H).

A Accel Byte Sets the driving speed as an 8-bit binary value.

S Status Byte

The upper 3 bits are the MSB of the steering angle, and the lower 5 bits are the status bits. Push button switches are assigned to these bits.

h Checksum Value

Exclusive OR value of θ, A, and S

* Terminator 0x2A

For information about the optical remote control, see section 2.6 Remote Control Receiver Module.

(2) Automatic steering angle order The steering angle during automatic driving is calculated from the information from the tilt sensor, turning sensor, and speed sensor, as well as the steering angle information from the remote control.

(3) Comparison calculation

Within the automatic handle AUT_HDL() or manual handle HND_HDL(), deviation = (handle order angle) - (feedback angle) is calculated. With the deviation as an argument, if the order angle is large, the handle jumps to the left rotation L_side(), and if the order angle is small, the handle jumps to the right rotation R_side(), thereby setting up proportional control.

(4)Motor Driver

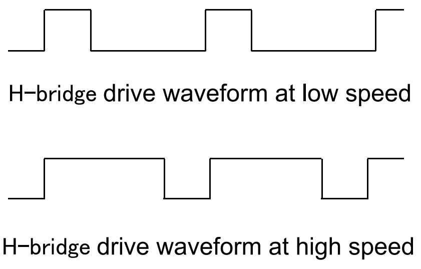

H-bridge drive waveform at low speed
H-bridge drive waveform at high speed

Figure 5.5 Speed control by PWM

The hardware drives the steering motor with a PWM-controlled H -bridge proportional to the deviation.

The software uses the timer's square wave output mode as shown in Figure 5 .5 with the L_side( ) or R_side( ) settings, drives the H-bridge with a pulse width proportional to the deviation, and performs proportional control by varying the motor speed. For information on PWM control and the H-bridge driver, see Section 2.4.

(5)Rotary Encoder

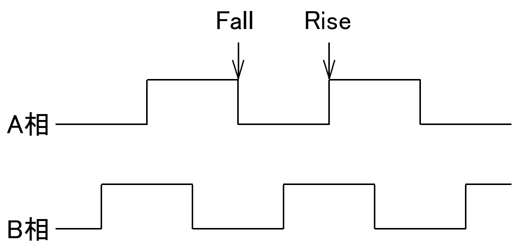


Figure 5.6 Encoder output waveform

The bicycle teaching materials use an AB-phase incremental type rotary encoder, whose output waveform is as shown in Figure 5.6, to detect feedback of the handlebar angle.

The handle angle encoder is explained in Section 1.2.2. Please refer to it. This encoder is built into the handle drive motor and generates 12 pulses per rotation. When calculated from the motor reduction ratio, 507.7 pulses are generated for a handle angle of ±70°.

In this bicycle teaching material , the rotation angle is detected at the falling and rising edges of the A phase as shown in Figure 5.6 to further improve the resolution, so 1015.4 pulses (±507. 7 pulses) are obtained for a handle angle of ±70°, enabling smooth handle control. Edge detection is performed by an interrupt, and the encoder's pulse buffer Hndl_feedback or (1070H) is incremented or decremented according to the direction at each interrupt. The interrupt function name is as follows:

TLCS－900
Falling interrupt → INT5_Z(), rising edge interrupt→ INT6_Z()

ARM
Falling interrupt → INT1_IRQHandler(), rising interrupt → INT2_IRQHandler()

5.2.2 ON-OFF control and proportional control

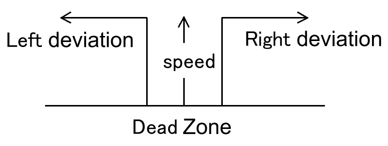


Figure 5.7 ON/OFF control

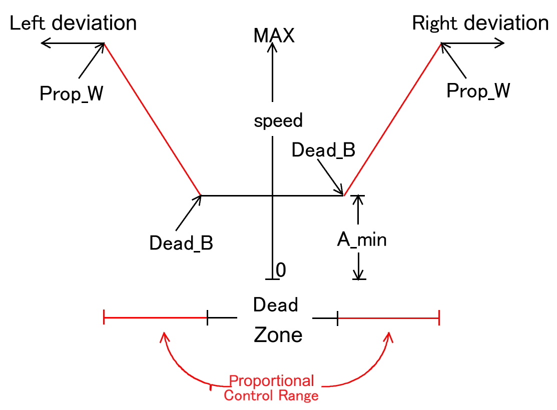


Figure 5.8 Proportional control

Proportional control, such as electric servo mechanisms and proportional solenoid valves, is commonly used in the mechanical control of industrial equipment.

There are two types of actuator drive: ON -OFF control and proportional control. In ON - OFF control, as shown in Figure 5.7, when the deviation between the order and feedback

exceeds the deadband width, ① the actuator turns 'ON' and moves at maximum speed in the direction that reduces the deviation. When the deviation amount enters the deadband width, the actuator turns 'OFF'.

When this ON-OFF operation is performed with an electric actuator, for example, ② the deviation enters the deadband and the motor is turned 'OFF', but the motor cannot stop immediately, so the actuator runs for a while and then stops.

If the deadband width is narrow, there is a possibility that it will run beyond the outside of the deadband on the other side, in which case ① the actuator turns 'ON' again, moves at maximum speed in the direction that reduces the deviation, and when the deviation amount enters the deadband width, the actuator turns 'OFF'.

If the dead band is set narrow with ON-OFF control, the above steps ① and ② will be repeated, resulting in a symptom known as hunting, and the actuator will go back and forth and will not stop.
Since the dead band cannot be made very narrow with ON-OFF control, you cannot expect very good stopping position accuracy.
With proportional control, the actuator decelerates in proportion to the deviation, as shown in Figure 5.8, so "overrunning" does not occur much and the stopping position accuracy is good.

5.2.3 Proportional Control

Proportional control performs mechanical control with an actuator speed proportional to the deviation between order and feedback, as shown in Figure 5.8. Proportional control speeds are divided into 1) the deadband range where the actuator does not move, 2) the proportional control range where the deviation and speed are proportional, and 3) the control range at maximum speed. 1) Deadband, deadband Even when proportional control is performed, a minimum deadband is necessary. Set a deadband that ensures a stable state without fine hunting.

If there is backlash or play in the mechanical parts or a time delay in the feedback system, problems such as hunting that does not stop or unstable stopping position accuracy will occur, so it is important to investigate the cause. Also, even if the actuator startup speed is set from zero, there is a high possibility that the actuator will not move because there is not much startup torque in reality.

It is necessary to set a minimum speed at startup, such as A_min in Figure 5.8. ② Proportional control range This is the region that accelerates and decelerates from the minimum speed to the maximum speed with a certain deviation width, and the purpose is to smoothly start and stop the actuator. If this region is shortened, it will be equivalent to ON-OFF control, and if it is longer, the actuator's tracking speed will be slower. ③ High- speed control range This is the region where the actuator operates at high speed. The maximum speed value can also be adjusted. The names of the proportional control functions are as follows:

```
TLCS－900
AUT_HDL( )  )  Calculate the following direction  ( L_side( )
HND_HDL( )  )  and deviation and jump             ( R_side( )
    ↓
L_side( )   )  Dead band of proportional control Dead_B, proportional control width Prop_W
R_side( )   )  After setting the minimum speed A_min, set the timer PWM


ARM
AUT_HDL( )  )                                      Hensa_check( )
            )  Calculate deviation and jump
HND_HDL( )  )
    ↓
Hensa_check( )  Calculate the tracking direction from the deviation, and when it is
    ↓           greater than the dead band width L_turn( )
                                                 R _turn( )

L_turn( )   )  Motor rotation control
R_turn( )   )
```

5.3 Pedal control

Pedal control (vehicle speed control) is performed by remote control to control forward and reverse, and speed control by PWM. In addition , the vehicle speed is measured by counting pulses from the rotary encoder built into the motor, but the encoder value is not fed back like in steering wheel control, so it is an open loop.

5.3.1 Pedal Control Overview

As shown above in Figure 5.4 Remote Control Message, an 8-bit speed command is sent from the remote control. The 8-bit value 0 to 255 is divided into three parts, forward (Ahead), stop, and reverse (Asturn), and you can freely control forward and backward movement using the accelerator knob on the remote control. The pedal is operated as follows:

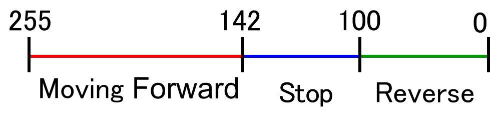


Figure 5.9
Speed command from remote controller

HND_PDL( ) — Manual
AUT_PDL( ) — Automatic

The pedal order shown in Figure 5.9 is generated based on the current accelerator position of the remote control.

Advance Order AHpedal( )
Backward Order ASpedal( )

If the pedal order is forward , depending on the current situation, it will start forward, stop to reverse, or maintain the current situation and only set the speed.

AHpedal( )

Motor stopped — Yes → Motor forward start
↓
Motor reversing — Yes → Motor reversal preparation/stop
↓
Motor moving forward — Yes → Speed setting update

When the pedal order is reverse, just like forward, you stop or maintain the current state to start reverse or reverse depending on the current situation, and only set the speed.

ASpedal( )

Motor stopped — Yes → Motor reverse start
↓
Motor moving forward — Yes → Motor reversal preparation/stop
↓
Motor reversing — Yes → Speed setting update

5.3.2 Pedal Speed settings

The square wave output of the CPU's built-in timer is connected to the PWM terminal of the H-bridge driver that drives the pedal motor for PWM control.

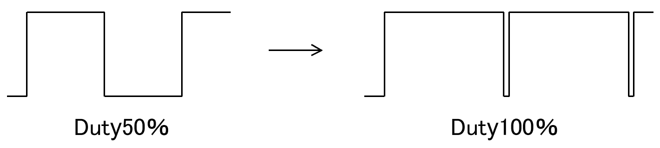


Figure 5.10 Timer square wave output waveform

The duty ratio of the timer square wave output can be varied as desired by a program, as shown in Figure 5.10. However, even if the duty ratio is set to its maximum, for example, in the case of an 8- bit timer, the duty ratio will be up to 254/255.

In the TLCS-900 version example program, when AHrning() detects a duty of 254/255 or more during forward rotation only, the timer output is switched to the boat output and '1' is output continuously , making it 255/255.

※The same process is carried out for steering wheel speed control.

5.3.3 Pedal motor speed measurement

The traveling speed of a bicycle is one of the essential elements in bicycle posture control calculations. In this teaching material, the traveling speed is measured by counting the number of output pulses from the rotary encoder built into the pedal motor at regular intervals. For specifications of the pedal motor and rotary encoder, refer to section 1.3 above. Pulse measurement is performed by counting the number of pulses at 100 ms intervals in the interrupt function shown below, and then storing the result in Pedl_encoder or Pedl_enc or (1040H) after scaling the number of pulses x 15 to make it a convenient scale for posture control calculations. The number of pulses after scaling is a value of approximately 450 to 1100. Pulse count interrupt function name
T LCS－900 → INT0_Z( )
A RM → INT0_IRQHandler( )

5.4 Automatic Attitude Control

The automatic attitude control of the bicycle teaching material in Figure 5.11 operates the handlebars to return the bike to its original tilt, just like when we normally ride a bike, and also operates the same handlebars to turn in the desired direction. In this section, we will first explain the mechanism of attitude control, and in the next section we will explain how to develop an attitude control program.

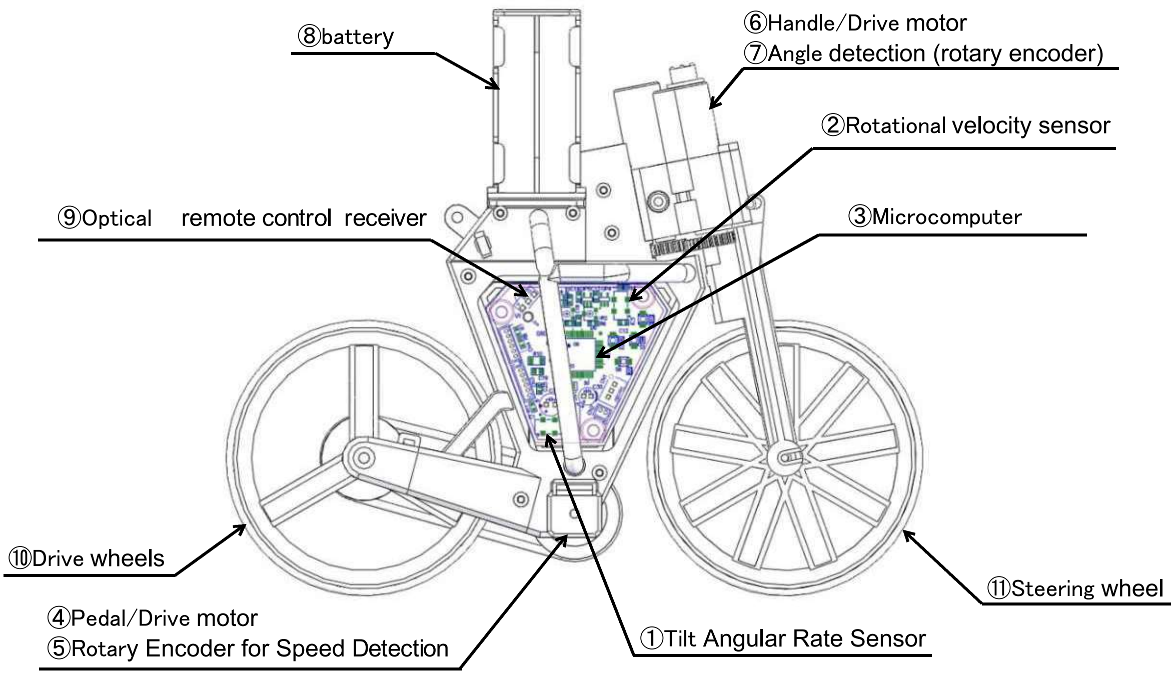

⑧battery
⑥Handle/Drive motor
⑦Angle detection (rotary encoder)
②Rotational velocity sensor
⑨Optical remote control receiver
③Microcomputer
⑩Drive wheels
⑪Steering wheel
④Pedal/Drive motor
⑤Rotary Encoder for Speed Detection
①Tilt Angular Rate Sensor

Figure 5.11 Automatic attitude control bicycle teaching material

5.4.1 Forces acting on a bicycle

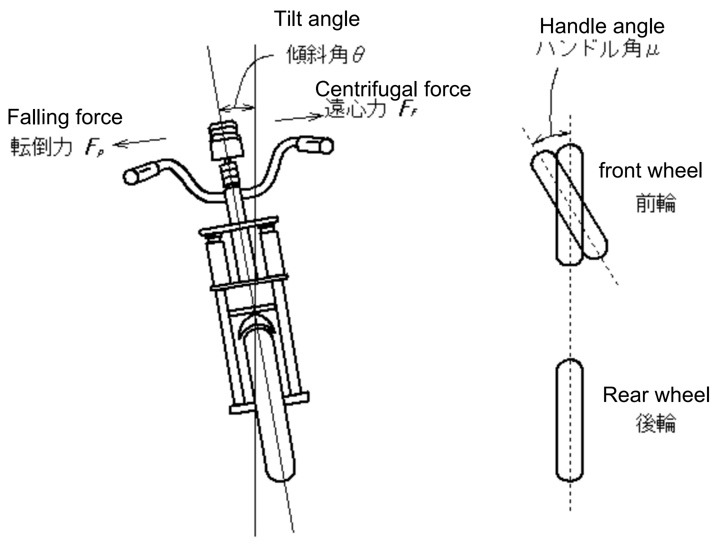


Figure 5.12 Forces acting on a bicycle

While a bicycle is being ridden, there are various forces at work, such as the tipping force F_P generated by the tilt of the body, the centrifugal force F_f generated when the handlebars are turned and the body turns, and the inertial force F_i that tries to keep the body in place as only the front wheel moves toward the steering side when the handlebars are turned.
In addition to these forces, there are various other forces at work, such as the force due to acceleration generated at the moment the handlebars are turned, and the force due to lateral acceleration generated by increasing or decreasing the vehicle speed while the body is turning.

Here we will explain posture control using the major forces of inversion, centrifugal force, and inertia.
As shown in Figure 5.12, the vehicle body inclination angle θ generates a force F_P that tries to tip the vehicle body over, as shown in Equation 5.1.

Falling force $F_{p} = mg \tan\theta$ 〔N〕 ･･･Equation 5.1

θ･･･Tilt angle m･･･Mass at center of gravity

Next, the turning radius r when the vehicle turns at a steering angle μ is affected by the wheelbase length and can be expressed by Equation 5.2.

Turning radius $r = \frac{K_1}{\mu}$ 〔m〕･･･Equation 5.2

μ･･･Handle angle K_1･･･Constants affected by wheelbase length

When the vehicle speed is V 〔m/s〕, the centrifugal force F_f is given by equation 5.3.

Centrifugal force $F_f = m\frac{V^2}{r}$ 〔N〕･･･Equation 5.3

Substituting Equation 5.2,

$$F_f = m\frac{V^2}{\frac{K_1}{\mu}} = \frac{1}{K_1} mV^2\mu \text{ 〔N〕･･･Equation 5.4}$$

m･･･Mass at center of gravity r･･･Turning radius

The other element, inertia force, is a force that causes the front wheel position to displace to the left in the right diagram of Figure 5.12 when turning due to the steering wheel angle, and tends to keep the vehicle in place, as shown in Equation 5.5.

Inertial force $F_i = K_2 mV\mu$ 〔N〕 ･･･Equation 5.5
m･･･Center of gravity mass near the front wheels

If the vector result of the tipping force, centrifugal force, and inertial force is zero, as in Equation 5.6, the bicycle will not fall over.

Overturning force - centrifugal force - inertial force = 0

$$mg\tan\theta - \frac{1}{K_1} mV^2\mu - K_2 mV\mu = 0 \cdots \text{Equation } 5.6$$

The steering angle μ can be derived from equation 5.6 as follows:

$$\mu = \frac{mg\tan\theta}{\frac{1}{K_1} mV^2 + K_2 mV} \cdots \text{Equation } 5.7$$

Since K_1 ・ K_2 ・ m are fixed values once the vehicle body is completed, if we combine them into a single constant K and consider only the dimensions, we get equation 5.8.

$$\mu = K\frac{\tan\theta}{V_2 + V} \cdots \text{Equation } 5.8$$

In the control program used in this teaching material, the speed V derived from the inertia force term is omitted, and since the angle of $\tan\theta$ is also small, the calculation is done at $\tan\theta = \theta$ using equation 5.9, and PID processing is performed.

$$\mu = K\frac{\theta}{V^2} \cdots \text{Equation } 5.9$$

The term for speed V^2 in equation 5.9 is very important, and the bicycle will not run properly unless the amount of handlebar operation is adjusted in response to changes in speed.

Column 5.1 Reversing a bicycle

Two-wheeled vehicles are not good at reversing.

$$mg\tan\theta - \frac{1}{K_1} mV^2\mu - K_2 mV\mu = 0 \cdots \text{Equation } 5.6$$ Vector of force acting when moving forward

$$mg\tan\theta - \frac{1}{K_1} mV^2\mu + K_2 mV\mu = 0 \cdots \text{Equation } 5.10$$ Force vector acting when reversing

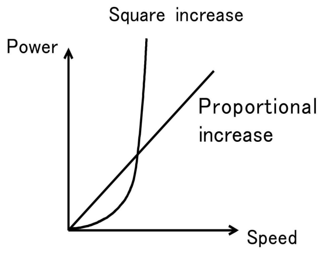


Figure 5.A: Squared increase and proportional increase

When moving forward, the directions of the centrifugal force and the inertial force are the same, but whe moving backward, the front wheel position in the right diagram of Figure 5.12 is displaced to the right. As result, the force of inertia is opposite the centrifugal force, as shown in Equation 5.10, and the two are cancelled out, weakening the restoring force trying to return the vehicle to its original position. Also, since the velocity term of the centrifugal force increases squarel and the velocity term of the inertial force is V, there comes a moment when the two forces completel at V^2 cancel each other out, at which point the restoring force becomes zero, and at the next moment the direction o the restoring force reverses, making it extremel difficult to steer when reversing.

5.4.2 Straightness correction

The bicycle can be operated without falling over by handling the handlebars according to equation 5.8 or 5.9, which is derived from the previous equation: tipping force - centrifugal force - inertia force = 0.
However, errors due to offset and drift of the inclination angular velocity sensor (① in Figure 5.11), which measures the inclination angle, an element of posture control, accumulate, making it difficult to ride in a straight line, especially for long periods of time.
To improve this straight-line riding ability, a turning angular velocity sensor (②) is used to correct the error of the inclination sensor (①).
Specifically, straight-line stability is improved by correcting the inclination angle in equation 5.12 using correction value C, which is obtained by proportional and integral processing of the output of the ② turning angular velocity sensor calculated in equation 5.11, and calculating the steering angle.

ω Turning correction value $C = K_3\omega + K_4\int \omega dt$ ···Equation 5.11

Handle angle $\mu = K\dfrac{\theta - \omega\text{Correction value}}{V^2}$ ···Equation 5.12

5.4.3 Control using a remote control

The direction of travel of an automatic attitude control bicycle can be controlled as desired using a remote control.
②The value of the turning correction value C, which is obtained by proportional integral processing of the output of the turning angular velocity sensor, is intentionally changed using an external remote control, and false turning information is given to equation 5.11, resulting in the bicycle turning.
When turning, the integral term of the turning correction value is cut. If this integral cut is not performed, straight-line stability after turning will be impaired.
The timing of this integral cut and how the accumulated error contained in the integral value is handled have a significant effect on the riding performance of the bicycle, and are one of the techniques in programming automatic attitude control.

5.5 Attitude control program

The automatic attitude control program explained in Section 5.4 is deployed in CALCU( ).
The output values of 0 to 1023 from the 10-bit A/D conversion, which is performed once every 0.313 ms, are accumulated eight times, and data from 0 to 8184 is prepared once approximately every 2.5 ms, then CALCU( ) is executed and the following five calculation functions are called.
When explaining the contents of the functions, the input and output variable names are listed in the order of the TLCS-900 C version, ARM C version, and TLCS-900 ASM

5.5.1 V2_SCAL( )

The actual speed pulse value Pedl_encoder or Pedal_enc or (1040H) is squared and scaled so that the output value is approximately 40 to 1000, then output to Speed_sq or (1052H) and used by RUDDER() to calculate the steering angle for autonomous

5.5.2 TRNrate( )

Calculates the average and moving average of the turning value.
The turning value input Turn_val_tmp or Turn_P_buf or (104AH) is input once every 2.5 ms. This is accumulated 12 times to obtain polarized int data once every 30 ms, which is then put into eight ring buffers to calculate the moving average.
The moving average output of the turning angular velocity Tangle_ave or (10A4H) is output once every 30 ms and is used within TurnPID() to determine the limit of the steering angle.

5.5.3 TurnPID( )

To improve the straight-line stability of the bicycle, the lean angle error shown in Equation 5.12 is corrected using the correction value Equation 5.11.
The output of the proportional term of the correction value is Turn_Val or Turn_P_calc or (1046H).
The output of the integral term of the correction value is Turn_I_calc or (104EH).
When integration is used, the integration value limit and reset timing are difficult elements.
In the current program, turning integration is stopped while turning.
The integration limit is set to a maximum limit of ±1000000 bits within K_max().
To prevent the sensor drift value from accumulating, one bit is subtracted once every 2.5 ms.
Although we are taking the above measures, we believe that the current situation is not optimal.

ω Turning correction value $C = K_3\omega + K_4\int \omega dt$ ・・・Equation 5.11

Handle angle $\mu = K\dfrac{\theta - \omega\text{Correction value}}{V^2}$ ・・・Equation 5.12

To control the bicycle using a remote control, the value of K3 in Equation 5.11, the turning compensation value, is intentionally changed by the handlebar angle on the remote control.
When the bicycle is in automatic driving mode, a dead band of ±200 bits is set around the center of the handlebar angle on the remote control to distinguish between turning and going straight.
Within ±200 bits, the bicycle will go straight, and any handlebar angle greater than this will cause the bicycle to jump to LtrnODR( ) or RtrnODR
( ) and enter turning mode, where the bicycle will start turning in the desired direction.
If the turning compensation value is changed significantly all at once from the remote control, the bicycle will fall over, so a limit is set so that the change value is no more than 50 bits in LtrnLIM( ) or RtrnLIM( ) once every 0.5 seconds.

5.4.4 KsyaPID

In the above section 5.4.1, $\mu = K\dfrac{\theta}{V^2}$ ・・・formula 5.9 is explained as PID processing, but there is no specific program that performs PID calculations.
We will explain the actual state of PID processing of the lean angle of this bicycle.
As explained above in Section 2.1, the angular velocity sensor section, and Figure 2.3, Sensor output waveform, the lean angle sensor of this bicycle has an output

waveform that is somewhere between that of an angular velocity sensor and an angular acceleration sensor, so if we think of it as an inclination angle sensor, the A/D converted output of the lean angle sensor, AD2_out_slope, AIN5_auto, or (102CH), is close to the differential value D of the lean angle.
Also, since Equation 5.11 is the differential value of the actual amount of turning resulting from the handlebar angle calculated from the differential value of the lean angle, , it is a proportional action in terms of dimensions, and can be thought of as an integral action.
When we first began developing this bicycle, we tried differentiating and integrating the lean angle, and then integrating it twice, and we arrived at this conclusion through trial and error.
KsyaPID() adds the above differential, proportional, and integral values, adjusts the span, and passes it to RUDDER() as the correction value Slope_Val or (1048H), which is the numerator term in equation 5.12.

5.5.5 RUDDER( )

Handle angle $\mu = K\dfrac{\theta - \omega \text{Correction value}}{V^2}$ ···Equation 5.12

A calculation is performed and proportional control of the steering angle is performed based on the deviation of (steering wheel angle - steering wheel feedback).

Chapter 6 Using the Debugging Function

You can use the debug function of the bicycle teaching material to record driving data during automatic driving. Normally, you can check the memory etc. with the debug tool connected, but you cannot continue to ride the bicycle with the tool connected. Therefore, you can temporarily record the driving data while driving to the memory, and then connect the tool after stopping the driving to view the recorded driving data. We will introduce how to record driving data using the sample program on the included CD.

6.1 Overview of how to record driving data

・Recording starts with the remote control UP button (LED: ON)
・Recording stops with the DOWN button on the remote control (LED: OFF)
・Recording interval: Approximately every 100 msec
・Recording capacity: Approximately 1 minute
・Recorded data ①: Steering wheel operation amount
・Recorded data ②: Incline value during automatic operation

The remote control sends commands to start and stop recording to the bicycle while it is in automatic operation. The red and green buttons on the remote control have already been assigned functions, so use the buttons under the cover. To remove the cover, remove the two screws at the top of the back side (Photo 6.1). Then, lift the top cover on the front side and it can be easily removed. Of the four buttons under the cover, the UP button starts recording and the DOWN button stops recording. The UP and DOWN buttons are used for the dump function on the remote control board, but do not affect the signals sent to the vehicle.

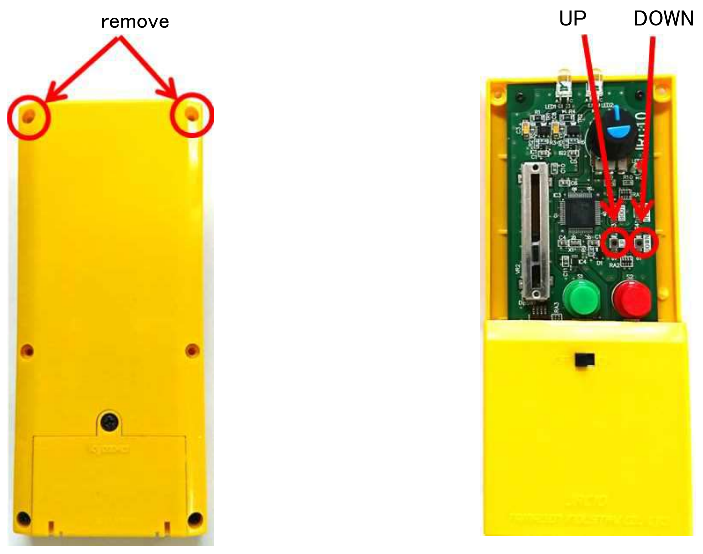
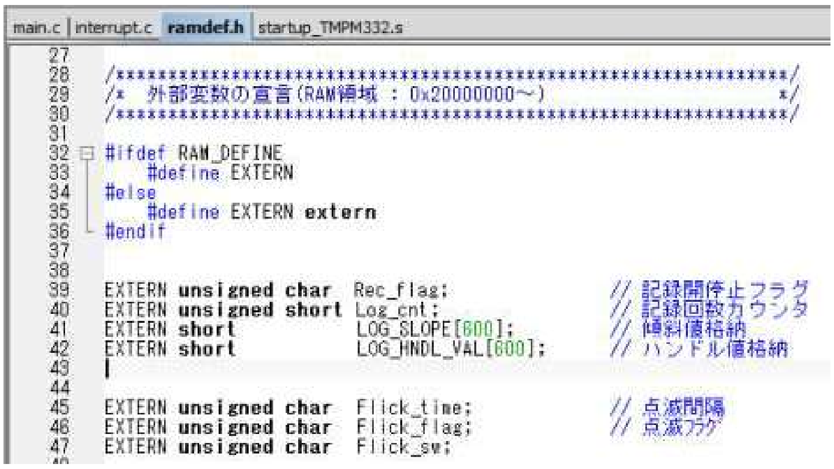


Photo 6.1 UP/DOWN button switch

6.2.1 ARM version debugging procedure1.

Refer to the "Setup (ARM version)" manual on the included CD and follow the steps up to "3.5 Writing and Debugging Executable Files." Leave the sample program written to the vehicle. 2. Write the variables for recording. Open the file "ramed.h" and add four variables by referring to image 6.1.

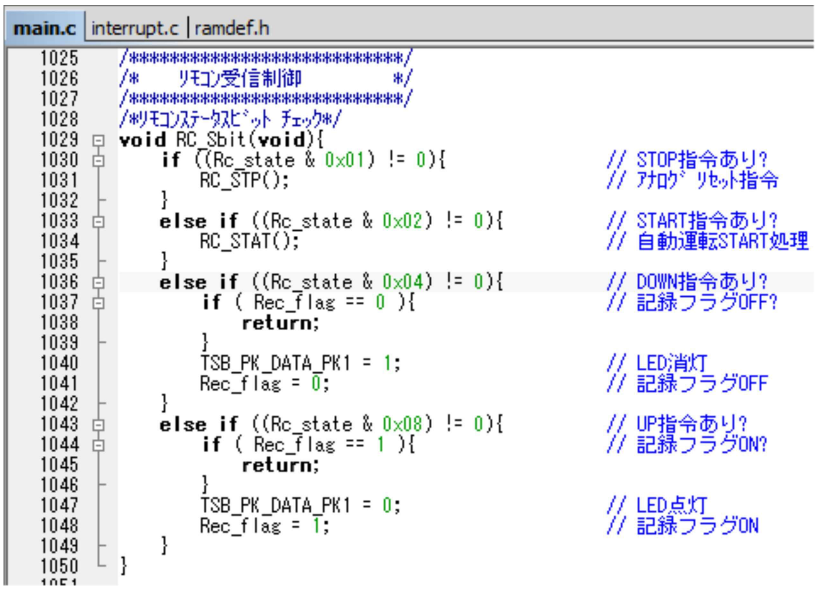


```
/***********************************************************************/
/*  外部変数の宣言(RAM領域 : 0x20000000~)                                 */
/***********************************************************************/

#ifdef RAM_DEFINE
    #define EXTERN
#else
    #define EXTERN extern
#endif


EXTERN unsigned char  Rec_flag;             // 記録開停止フラグ
EXTERN unsigned short Log_cnt;              // 記録回数カウンタ
EXTERN short          LOG_SLOPE[600];       // 傾斜値格納
EXTERN short          LOG_HNDL_VAL[600];    // ハンドル値格納


EXTERN unsigned char  Flick_time;           // 点滅間隔
EXTERN unsigned char  Flick_flag;           // 点滅ﾌﾗｸﾞ
EXTERN unsigned char  Flick_sw;
```

Image 6.1 Adding variables (ARM)

③ Write a program for recording.

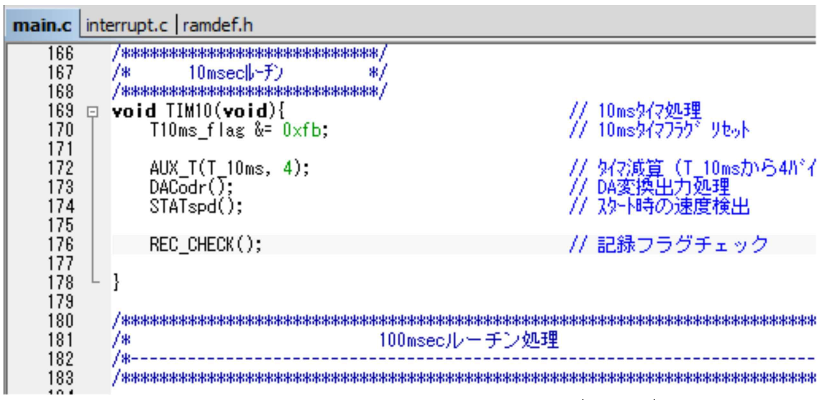
Open the file "main.c" and write flag control in the DOWN command in the function 'RC_Sbit( )' and in the UP command (Image 6.2). The DOWN command turns the flag OFF, and the UP command turns the flag ON and the LED on the car body ON.



```
/*****************************/
/*    ﾘﾓｺﾝ受信制御           */
/*****************************/
/*ﾘﾓｺﾝｽﾃｰﾀｽﾋﾞｯﾄ ﾁｪｯｸ*/
void RC_Sbit(void){
    if ((Rc_state & 0x01) != 0){                    // STOP指令あり?
        RC_STP();                                   // ｱﾅﾛｸﾞ ﾘｾｯﾄ指令
    }
    else if ((Rc_state & 0x02) != 0){               // START指令あり?
        RC_STAT();                                  // 自動運転START処理
    }
    else if ((Rc_state & 0x04) != 0){               // DOWN指令あり?
        if ( Rec_flag == 0 ){                       // 記録フラグOFF?
            return;
        }
        TSB_PK_DATA_PK1 = 1;                        // LED消灯
        Rec_flag = 0;                               // 記録フラグOFF
    }
    else if ((Rc_state & 0x08) != 0){               // UP指令あり?
        if ( Rec_flag == 1 ){                       // 記録フラグON?
            return;
        }
        TSB_PK_DATA_PK1 = 0;                        // LED点灯
        Rec_flag = 1;                               // 記録フラグON
    }
}
```

Image 6.2 Flag control (ARM)

④ Write a call to the recording flag check function 'REC_CHECK( )' in the function 'TIM10( )' (Image 6.3). The function called in the function 'TIM10( )' is executed every 10 msec.

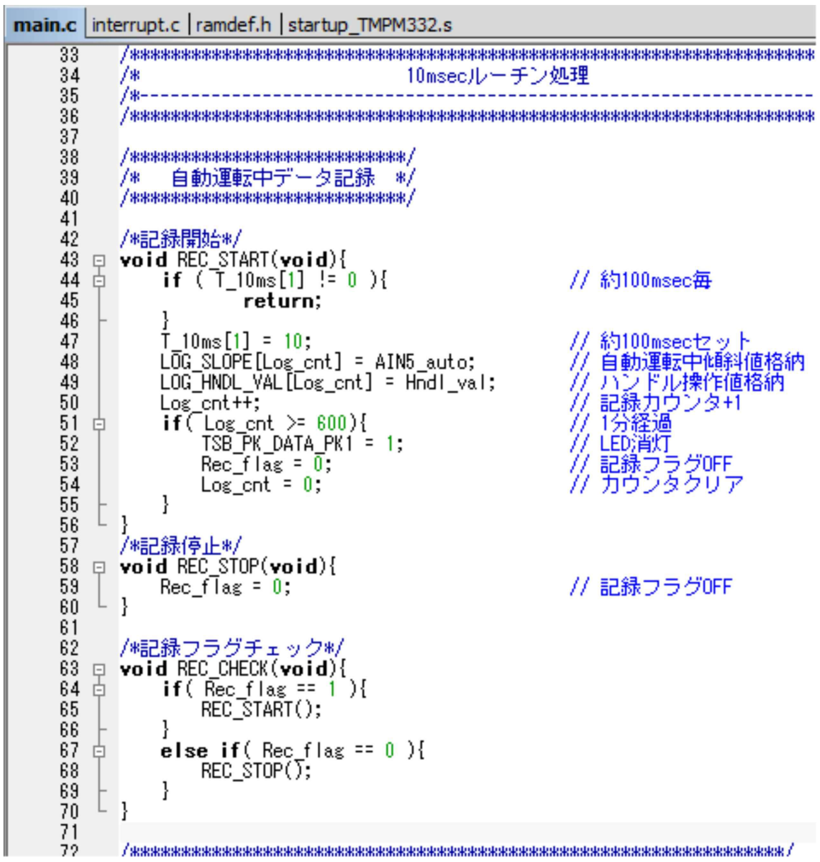


main.c | interrupt.c | ramdef.h

```
/****************************/
/*       10msecﾙｰﾁﾝ          */
/****************************/
void TIM10(void){                                      // 10msﾀｲﾏ処理
    T10ms_flag &= 0xfb;                                // 10msﾀｲﾏﾌﾗｸﾞ ﾘｾｯﾄ

    AUX_T(T_10ms, 4);                                  // ﾀｲﾏ減算 (T_10msから4ﾊﾞｲ
    DACodr();                                          // DA変換出力処理
    STATspd();                                         // ｽﾀｰﾄ時の速度検出

    REC_CHECK();                                       // 記録フラグチェック

}

/************************************************************************
/*                          100msecルーチン処理
/*----------------------------------------------------------------------
/************************************************************************
```

Image 6.310ms call (ARM)

⑤ Write the functions 'REC_START( )', 'REC_STOP( )', and 'REC_CHECK( )' below the comments for the 10msec routine processing (Image 6.4).



main.c | interrupt.c | ramdef.h | startup_TMPM332.s

```
/************************************************************************
/*                          10msecルーチン処理
/*----------------------------------------------------------------------
/************************************************************************

/****************************/
/*    自動運転中データ記録   */
/****************************/

/*記録開始*/
void REC_START(void){
    if ( T_10ms[1] != 0 ){                             // 約100msec毎
            return;
    }
    T_10ms[1] = 10;                                    // 約100msecセット
    LOG_SLOPE[Log_cnt] = AIN5_auto;                    // 自動運転中傾斜値格納
    LOG_HNDL_VAL[Log_cnt] = Hndl_val;                  // ハンドル操作値格納
    Log_cnt++;                                         // 記録カウンタ+1
    if( Log_cnt >= 600){                               // 1分経過
        TSB_PK_DATA_PK1 = 1;                           // LED消灯
        Rec_flag = 0;                                  // 記録フラグOFF
        Log_cnt = 0;                                   // カウンタクリア
    }
}
/*記録停止*/
void REC_STOP(void){
    Rec_flag = 0;                                      // 記録フラグOFF
}

/*記録フラグチェック*/
void REC_CHECK(void){
    if( Rec_flag == 1 ){
        REC_START();
    }
    else if( Rec_flag == 0 ){
        REC_STOP();
    }
}

/**********************************************************************/
```

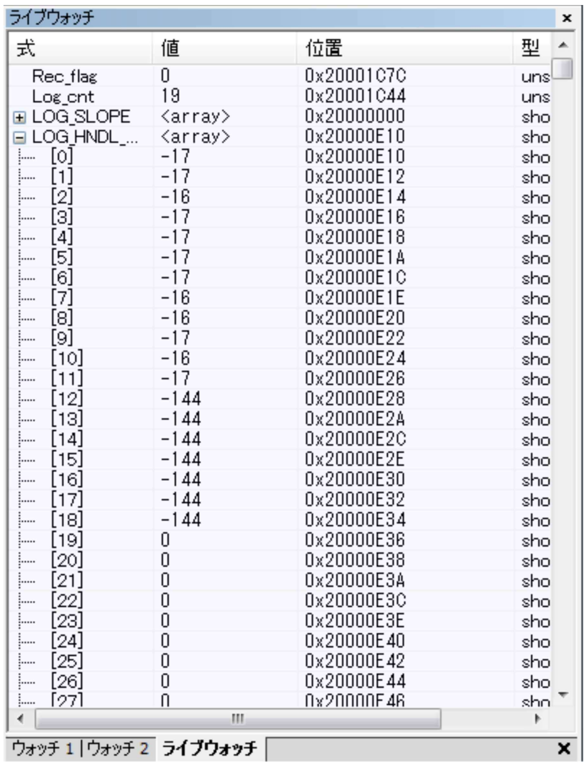
Image 6.4 Recording control (ARM)

The function 'REC_START( )' stores data every 100 msec.
AIN5_auto stored in the variable LOG_SLOPE[ ] is the slope value data during autonomous driving.
Hndl_val stored in LOG_HNDLE_VAL[ ] is the steering wheel operation amount data.

Data is stored approximately every 100 msec, up to 600 items, allowing one minute of data to be recorded. If recording is stopped with the DOWN button before one minute has elapsed, the data recorded up to the point at which it was stopped will be retained, and when recording is resumed, data will be stored from the point where it was stopped. After one minute has elapsed, the flag will automatically be turned OFF and recording will stop. If recording is started again, the oldest data will be overwritten.

⑥After writing the above program, rebuild everything and check that there are no errors. If there are no errors, connect the vehicle and the debug tool, download and debug.

⑦Check that the program is working properly on the debug screen. Display the live watch and register the four variables you added this time (Image 6.5). Press the UP button while the program is running to start recording. Use the handle volume on the remote control to change the value to be stored. Press the + button next to LOG_HNDLE_VAL[ ] to check the value. Press the DOWN button to stop storage and confirm that the LED on the car body turns off. Also, confirm that Log_cnt increases while recording and Rec_flag is 1.



式	値	位置	型
Rec_flag	0	0x20001C7C	uns
Log_cnt	19	0x20001C44	uns
⊞ LOG_SLOPE	<array>	0x20000000	sho
⊟ LOG_HNDL_...	<array>	0x20000E10	sho
[0]	-17	0x20000E10	sho
[1]	-17	0x20000E12	sho
[2]	-16	0x20000E14	sho
[3]	-17	0x20000E16	sho
[4]	-17	0x20000E18	sho
[5]	-17	0x20000E1A	sho
[6]	-17	0x20000E1C	sho
[7]	-16	0x20000E1E	sho
[8]	-16	0x20000E20	sho
[9]	-17	0x20000E22	sho
[10]	-16	0x20000E24	sho
[11]	-17	0x20000E26	sho
[12]	-144	0x20000E28	sho
[13]	-144	0x20000E2A	sho
[14]	-144	0x20000E2C	sho
[15]	-144	0x20000E2E	sho
[16]	-144	0x20000E30	sho
[17]	-144	0x20000E32	sho
[18]	-144	0x20000E34	sho
[19]	0	0x20000E36	sho
[20]	0	0x20000E38	sho
[21]	0	0x20000E3A	sho
[22]	0	0x20000E3C	sho
[23]	0	0x20000E3E	sho
[24]	0	0x20000E40	sho
[25]	0	0x20000E42	sho
[26]	0	0x20000E44	sho
[27]	0	0x20000E46	sho

Image 6.5 Live Watch

8. Once you have confirmed that the program is working properly, prepare the recorded data so that it can be easily handled. Select Break from the Debug screen to pause the program.

Display Watch 1 and 2 from the Display tab, and register LOG_HNDLE_VAL to Watch 1 and LOG_SLOPE to Watch 2 (Image 6.6) (Image 6.7).

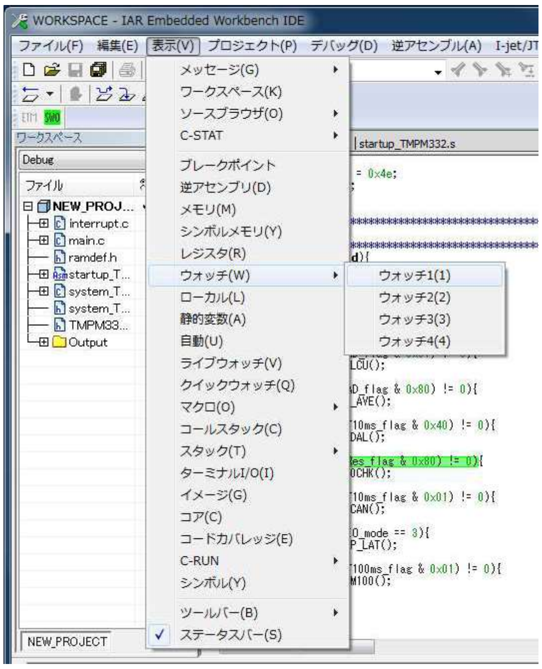
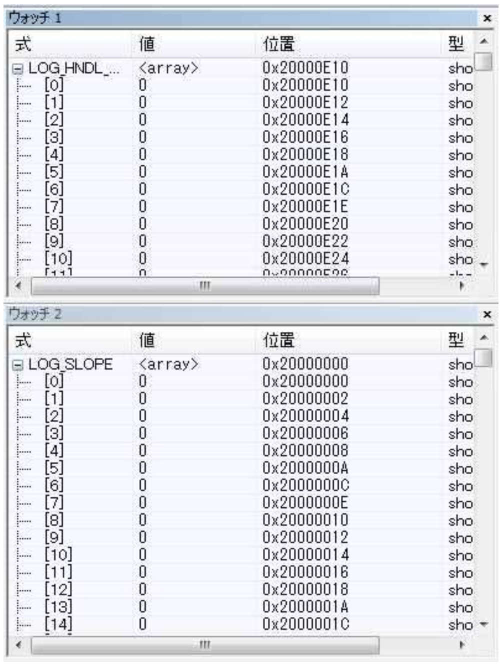


Figure 6.6 Watch display



ウォッチ 1

式	値	位置	型
LOG_HNDL_...	<array>	0x20000E10	sho
[0]	0	0x20000E10	sho
[1]	0	0x20000E12	sho
[2]	0	0x20000E14	sho
[3]	0	0x20000E16	sho
[4]	0	0x20000E18	sho
[5]	0	0x20000E1A	sho
[6]	0	0x20000E1C	sho
[7]	0	0x20000E1E	sho
[8]	0	0x20000E20	sho
[9]	0	0x20000E22	sho
[10]	0	0x20000E24	sho

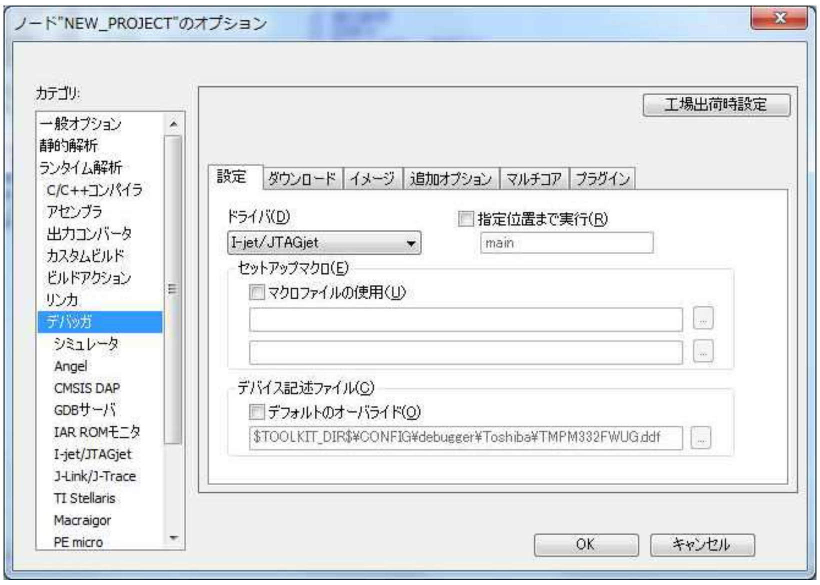
ウォッチ 2

式	値	位置	型
LOG_SLOPE	<array>	0x20000000	sho
[0]	0	0x20000000	sho
[1]	0	0x20000002	sho
[2]	0	0x20000004	sho
[3]	0	0x20000006	sho
[4]	0	0x20000008	sho
[5]	0	0x2000000A	sho
[6]	0	0x2000000C	sho
[7]	0	0x2000000E	sho
[8]	0	0x20000010	sho
[9]	0	0x20000012	sho
[10]	0	0x20000014	sho
[11]	0	0x20000016	sho
[12]	0	0x20000018	sho
[13]	0	0x2000001A	sho
[14]	0	0x2000001C	sho

Image 6.7 Watch Registration

Once registration is complete, stop debugging, turn off the power, and remove the tool. 9. Acquire data during automatic driving. Refer to the driving instruction video and perform automatic driving. Pressing the UP button just before driving will ensure reliable data recording. 10. After automatic driving has stopped, press the DOWN button to stop recording. Please note that the recorded values will be erased if the vehicle's power is turned off.

⑪ From the Project Options, select the Debugger Settings tab and uncheck Run to specified location (Image 6.8).



ノード"NEW_PROJECT"のオプション

カテゴリ: デバッガ

設定

ドライバ(D): I-jet/JTAGjet

☐ 指定位置まで実行(R): main

セットアップマクロ(E)

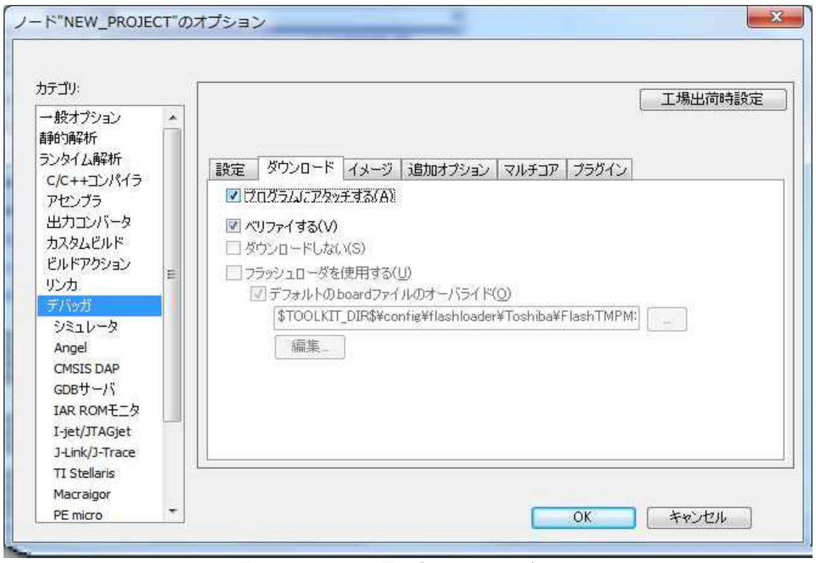
☐ マクロファイルの使用(U)

デバイス記述ファイル(C)

☐ デフォルトのオーバライド(O): $TOOLKIT_DIR$¥CONFIG¥debugger¥Toshiba¥TMPM332FWUG.ddf

Image 6.8 Debugging settings

Next, select the Download Debugger tab from the project options and check Attach to program (Image 6.9). Checking the above allows you to debug without resetting the target. After changing the options, rebuild everything.



ノード"NEW_PROJECT"のオプション

カテゴリ:

- 一般オプション
- 静的解析
- ランタイム解析
- C/C++コンパイラ
- アセンブラ
- 出力コンバータ
- カスタムビルド
- ビルドアクション
- リンカ
- デバッガ
- シミュレータ
- Angel
- CMSIS DAP
- GDBサーバ
- IAR ROMモニタ
- I-jet/JTAGjet
- J-Link/J-Trace
- TI Stellaris
- Macraigor
- PE micro

工場出荷時設定

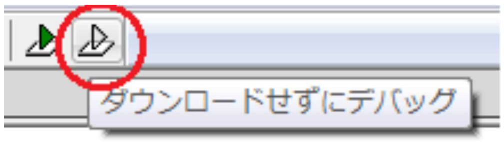
設定 | ダウンロード | イメージ | 追加オプション | マルチコア | プラグイン

- [✓] プログラムにアタッチする(A)
- [✓] ベリファイする(V)
- [ ] ダウンロードしない(S)
- [ ] フラッシュローダを使用する(U)
  - [✓] デフォルトのboardファイルのオーバライド(O)
  - $TOOLKIT_DIR$¥config¥flashloader¥Toshiba¥FlashTMPM:
  - 編集...

OK | キャンセル

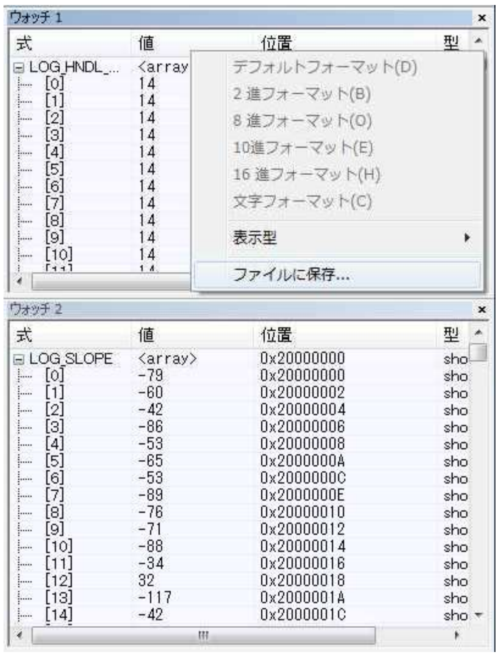
Image 6.9 Debug setting 2

⑫Connect the tool to the vehicle and perform debugging without downloading (Image 6.10).



画像6.10 デバッグ設定3

Right-click on the Watch 1 screen and select Save to file (Image 6.11).



ウォッチ 1

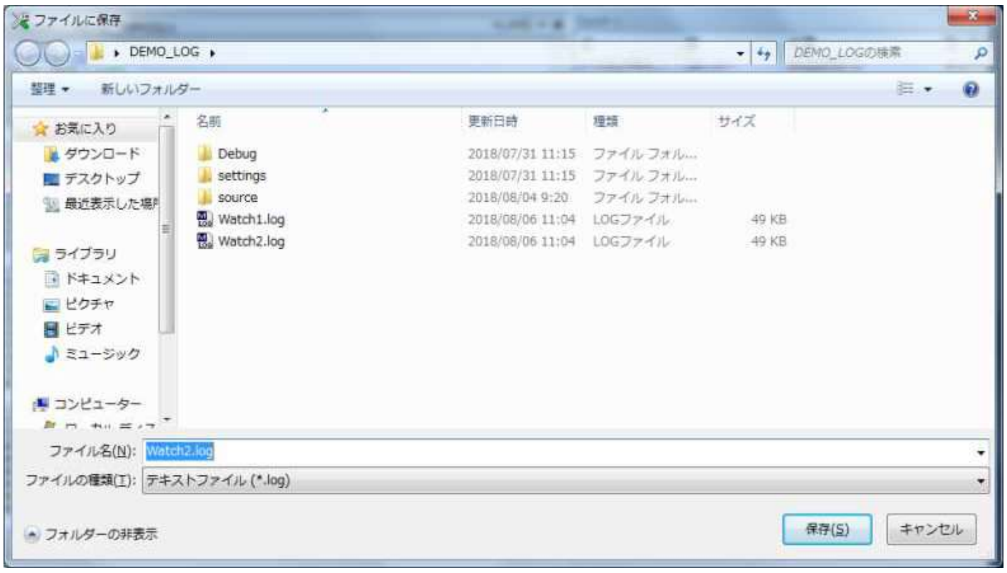
式	値	位置	型
LOG_HNDL_...	<array		
[0]	14		
[1]	14		
[2]	14		
[3]	14		
[4]	14		
[5]	14		
[6]	14		
[7]	14		
[8]	14		
[9]	14		
[10]	14		

ウォッチ 2

式	値	位置	型
LOG_SLOPE	<array>	0x20000000	sho
[0]	-79	0x20000000	sho
[1]	-60	0x20000002	sho
[2]	-42	0x20000004	sho
[3]	-86	0x20000006	sho
[4]	-53	0x20000008	sho
[5]	-65	0x2000000A	sho
[6]	-53	0x2000000C	sho
[7]	-89	0x2000000E	sho
[8]	-76	0x20000010	sho
[9]	-71	0x20000012	sho
[10]	-88	0x20000014	sho
[11]	-34	0x20000016	sho
[12]	32	0x20000018	sho
[13]	-117	0x2000001A	sho
[14]	-42	0x2000001C	sho

Image 6.11 Log save 1

Similarly, save the Watch 2 screen.
You can save it anywhere you like (Image 6.12).
If you have not taken a record, please refer to ⑭.


ファイルに保存

DEMO_LOG

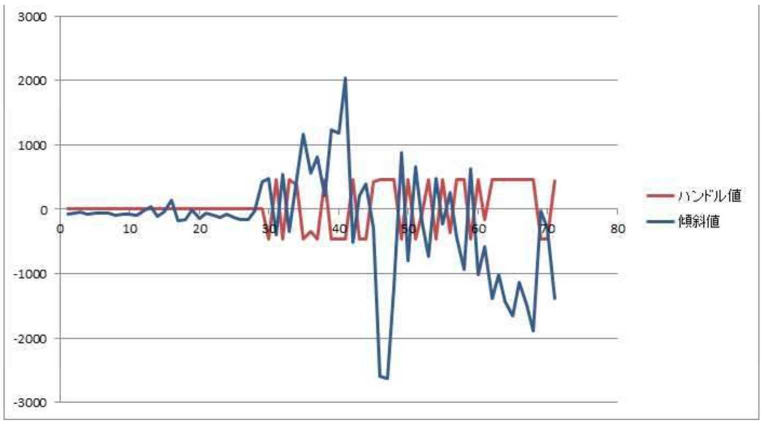
整理 ▾ 新しいフォルダー

名前	更新日時	種類	サイズ
Debug	2018/07/31 11:15	ファイル フォル...	
settings	2018/07/31 11:15	ファイル フォル...	
source	2018/08/04 9:20	ファイル フォル...	
Watch1.log	2018/08/06 11:04	LOGファイル	49 KB
Watch2.log	2018/08/06 11:04	LOGファイル	49 KB

ファイル名(N): Watch2.log
ファイルの種類(T): テキストファイル (*.log)

Image 6.12 Log save 2

⑬The saved log file can be edited using Excel etc. The acquired driving data is shown in Graph 6.1. This graph shows data every 100msec, but you can change the program to every 10msec or add turning values, pedal speed, etc. to the acquired data.



Graph 6.1 Driving record (ARM))

⑭ If no record is taken, please check steps ⑨ to ⑫ again. If no record is taken after checking, the vehicle board may have been reset. If you are using a J-TAG tool compatible with J-LINK, the target may be reset when connected.

Press the UP button, then press the DOWN button, and the LED on the vehicle will go off. If the LED goes on after connecting the debug tool with the LED off, the vehicle board has been reset.

Please take the following measures:

・After connecting the debug tool to the PC, wait until it is recognized by the PC before connecting it to the vehicle.

・If you are using a USB extension cable for your debugging tool, remove it.

・Replace the vehicle battery with a new one.

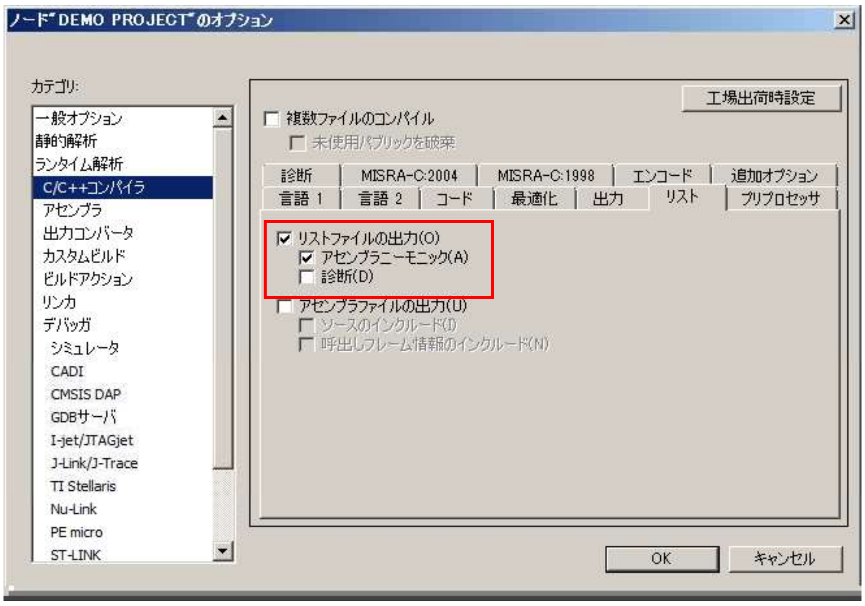
If the reset still occurs after trying the above, please use IAR's J-TAG tool I-JET or similar.

6.2.2 How to output assembler files in the ARM version

This explains how to output assembler files with IAR Embedded Workbench for Arm. Only the full version and the 30-day limited edition support outputting assembler files with IAR Embedded Workbench for Arm, and it cannot be used with the code capacity limited edition. If you have installed the code capacity limited edition, please install the 30-day limited edition from the IAR website. In this case, we have confirmed that it is possible to install the same IAR Embedded Workbench for Arm if the version is different. Below we will explain how to output assembly files with the ARM version.

Assembly file output settings

Open the relevant workspace and open the project options. Next, select the List tab in the C/C++ Compiler item. When you open the relevant tab, the screen below will be displayed. Currently, the "List file output" box is not checked so that the assembly file is not output. If you want to output an assembly file, please set it as shown below.



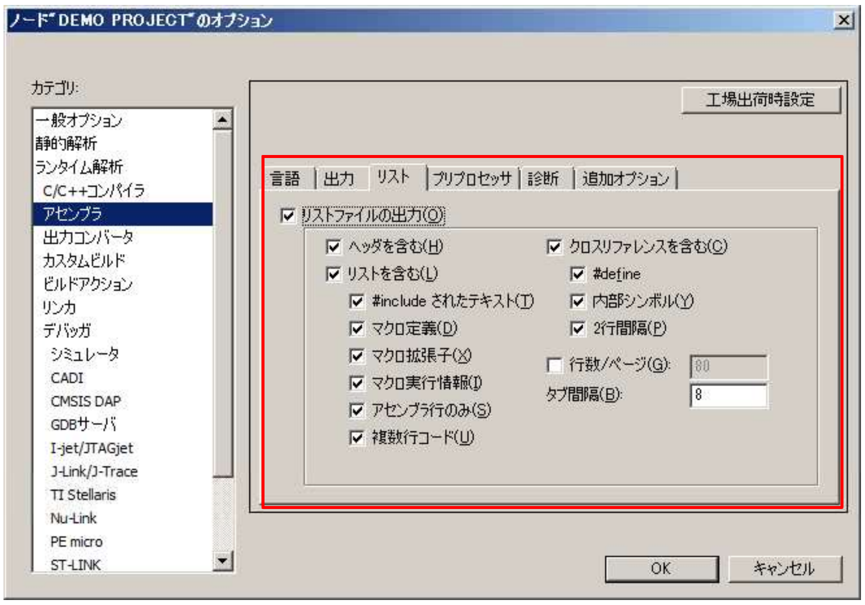
☐ 複数ファイルのコンパイル
☐ 未使用パブリックを破棄

☑ リストファイルの出力(O)
☑ アセンブラニーモニック(A)
☐ 診断(D)
☐ アセンブラファイルの出力(U)
☐ ソースのインクルード(I)
☐ 呼出しフレーム情報のインクルード(N)

Image 6.13: Project Options List Screen

Next, select the List tab for the Assembler item. When you open the tab, the following screen will be displayed. Currently, the "List file output" box is unchecked so that the assembly file is not output. To output the assembly file, see Image 6.14 on the next page.

Check the box for the assembly file output setting, click OK, and then rebuild. If there are no errors, the assembly file will be output to the List folder in the Debug folder where the project is saved, as shown in Image 6.15 Assembly File Status.


ノード"DEMO PROJECT"のオプション

カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
CADI
CMSIS DAP
GDBサーバ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Nu-Link
PE micro
ST-LINK

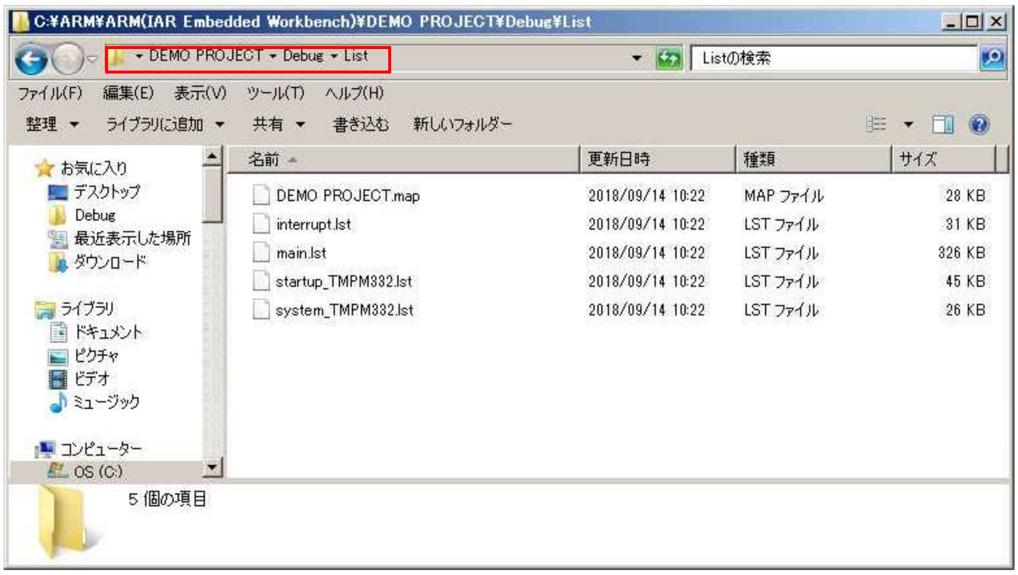
工場出荷時設定

言語 | 出力 | リスト | プリプロセッサ | 診断 | 追加オプション

☑ リストファイルの出力(O)
- ☑ ヘッダを含む(H)
- ☑ リストを含む(L)
  - ☑ #include されたテキスト(T)
  - ☑ マクロ定義(D)
  - ☑ マクロ拡張子(X)
  - ☑ マクロ実行情報(I)
  - ☑ アセンブラ行のみ(S)
  - ☑ 複数行コード(U)
- ☑ クロスリファレンスを含む(C)
  - ☑ #define
  - ☑ 内部シンボル(Y)
  - ☑ 2行間隔(P)
- ☐ 行数/ページ(G): 80
- タブ間隔(B): 8

OK　キャンセル

Image 6.14 Assembly file output settings


C:¥ARM¥ARM(IAR Embedded Workbench)¥DEMO PROJECT¥Debug¥List

DEMO PROJECT ▸ Debug ▸ List　　Listの検索

ファイル(F)　編集(E)　表示(V)　ツール(T)　ヘルプ(H)

整理 ▾　ライブラリに追加 ▾　共有 ▾　書き込む　新しいフォルダー

名前	更新日時	種類	サイズ
DEMO PROJECT.map	2018/09/14 10:22	MAP ファイル	28 KB
interrupt.lst	2018/09/14 10:22	LST ファイル	31 KB
main.lst	2018/09/14 10:22	LST ファイル	326 KB
startup_TMPM332.lst	2018/09/14 10:22	LST ファイル	45 KB
system_TMPM332.lst	2018/09/14 10:22	LST ファイル	26 KB

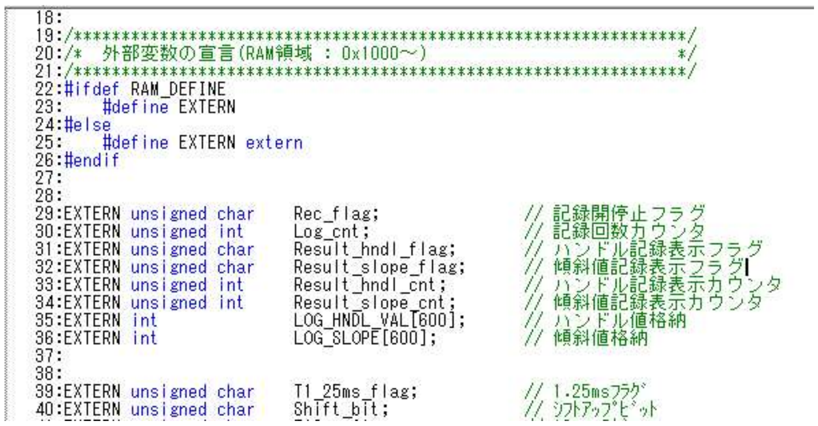
5 個の項目

Image 6.15 Assembly file status

If you open the above file in a text editor such as Notepad, you can see the program written in assembly language.

6.3 TLCS-900 version debugging procedure

① Refer to the manual "Setup (TLCS Edition)" on the included CD and execute up to "3.5 Writing the Executable File". Leave the sample program written to the vehicle. ② Write the variables for recording. Open the file "ramed.h" and add eight variables referring to image 6.16.

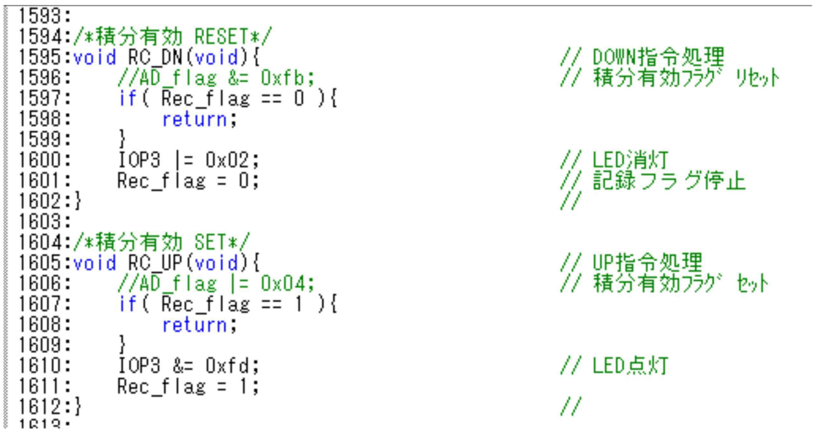


```
/*****************************************************************/
/*  外部変数の宣言(RAM領域 : 0x1000～)                              */
/*****************************************************************/
#ifdef RAM_DEFINE
    #define EXTERN
#else
    #define EXTERN extern
#endif


EXTERN unsigned char    Rec_flag;             // 記録開停止フラグ
EXTERN unsigned int     Log_cnt;              // 記録回数カウンタ
EXTERN unsigned char    Result_hndl_flag;     // ハンドル記録表示フラグ
EXTERN unsigned char    Result_slope_flag;    // 傾斜値記録表示フラグ
EXTERN unsigned int     Result_hndl_cnt;      // ハンドル記録表示カウンタ
EXTERN unsigned int     Result_slope_cnt;     // 傾斜値記録表示カウンタ
EXTERN int              LOG_HNDL_VAL[600];    // ハンドル値格納
EXTERN int              LOG_SLOPE[600];       // 傾斜値格納


EXTERN unsigned char    T1_25ms_flag;         // 1.25msﾌﾗｸﾞ
EXTERN unsigned char    Shift_bit;            // ｼﾌﾄｱｯﾌﾟﾋﾞｯﾄ
```

Image 6.16 Adding variables (900)

③ Write a program for recording.

Open the file "C_JTN.c" and write flag control in the control functions RC_DN() and RC_UP() for the remote control's DOWN command and UP command (Image 6.17). The DOWN command turns the flag OFF and turns off the LED, and the UP command turns the flag ON and turns on the car body's LED. Also, comment out the integral enable flag.

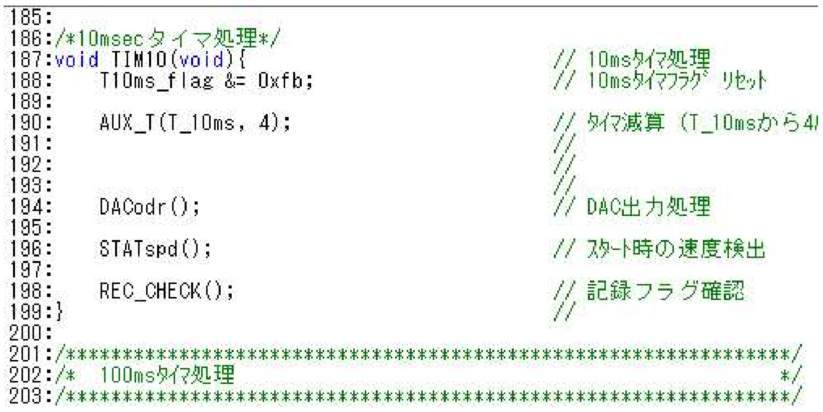


```

/*積分有効 RESET*/
void RC_DN(void){                                  // DOWN指令処理
    //AD_flag &= 0xfb;                             // 積分有効ﾌﾗｸﾞ ﾘｾｯﾄ
    if( Rec_flag == 0 ){
        return;
    }
    IOP3 |= 0x02;                                  // LED消灯
    Rec_flag = 0;                                  // 記録フラグ停止
}                                                  //

/*積分有効 SET*/
void RC_UP(void){                                  // UP指令処理
    //AD_flag |= 0x04;                             // 積分有効ﾌﾗｸﾞ ｾｯﾄ
    if( Rec_flag == 1 ){
        return;
    }
    IOP3 &= 0xfd;                                  // LED点灯
    Rec_flag = 1;
}                                                  //
```

Image 6.17 Flag control (900)

④ Write a call to the recording flag check function 'REC_CHECK( )' in the function 'TIM10( )'.

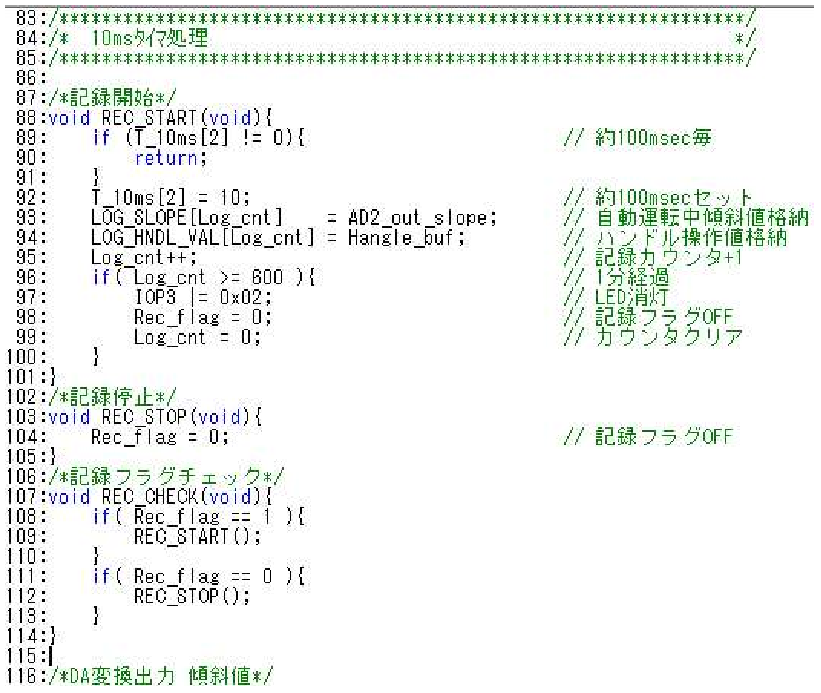
(Image 6.18). The function called within the function 'TIM10( )' will be executed every 10 ms.



```
/*10msecタイマ処理*/
void TIM10(void){                          // 10msﾀｲﾏ処理
    T10ms_flag &= 0xfb;                    // 10msﾀｲﾏﾌﾗｸﾞ ﾘｾｯﾄ

    AUX_T(T_10ms, 4);                      // ﾀｲﾏ減算 (T_10msから4
                                           //
                                           //
                                           //
    DACodr();                              // DAC出力処理

    STATspd();                             // ｽﾀｰﾄ時の速度検出

    REC_CHECK();                           // 記録フラグ確認
}                                          //

/*****************************************************************/
/*  100msﾀｲﾏ処理                                                  */
/*****************************************************************/
```

Image 6.18 10ms call (900)

⑤ Write the functions 'REC_START( )', 'REC_STOP( )', and 'REC_CHECK( )' below the comment for the 10msec timer processing (Image 6.19).



```
/*****************************************************************/
/*  10msﾀｲﾏ処理                                                   */
/*****************************************************************/

/*記録開始*/
void REC_START(void){
    if (T_10ms[2] != 0){                     // 約100msec毎
        return;
    }
    T_10ms[2] = 10;                          // 約100msecセット
    LOG_SLOPE[Log_cnt]    = AD2_out_slope;   // 自動運転中傾斜値格納
    LOG_HNDL_VAL[Log_cnt] = Hangle_buf;      // ハンドル操作値格納
    Log_cnt++;                               // 記録カウンタ+1
    if( Log_cnt >= 600 ){                    // 1分経過
        IOP3 |= 0x02;                        // LED消灯
        Rec_flag = 0;                        // 記録フラグOFF
        Log_cnt = 0;                         // カウンタクリア
    }
}
/*記録停止*/
void REC_STOP(void){
    Rec_flag = 0;                            // 記録フラグOFF
}
/*記録フラグチェック*/
void REC_CHECK(void){
    if( Rec_flag == 1 ){
        REC_START();
    }
    if( Rec_flag == 0 ){
        REC_STOP();
    }
}

/*DA変換出力 傾斜値*/
```

Image 6.19 Recording control (900)

The function 'REC_START( )' stores data every 100 msec. AD2_out_slope stored in the variable LOG_SLOPE[ ] is the slope value data during automatic driving. Hangle_buf stored in LOG_HNDLE_VAL[ ] is the handlebar operation amount data. Data is stored approximately every 100 msec, up to 600 items, so data can be recorded for one minute. If recording is stopped with the DOWN button switch before one minute has elapsed, the data recorded up to the point at which it was stopped will be retained, and when recording is resumed, data will be stored from the continuation. After one minute has elapsed, the flag will automatically be turned OFF and recording will stop. If recording is started again, the oldest data will be overwritten.

⑥Describe the conditions for displaying the data recording results. Press the UP button on the vehicle body board to display the handlebar recording value, and press the DN button on the vehicle body board to display the tilt recording value (Photo 6.2). Modify the sample program's dump function to include a flag for recording data (Image 6.20). Comment out the DUMP() call and the M_POINT change from the sample program. Also, pressing the same button while each result is being displayed will stop the display of the records.

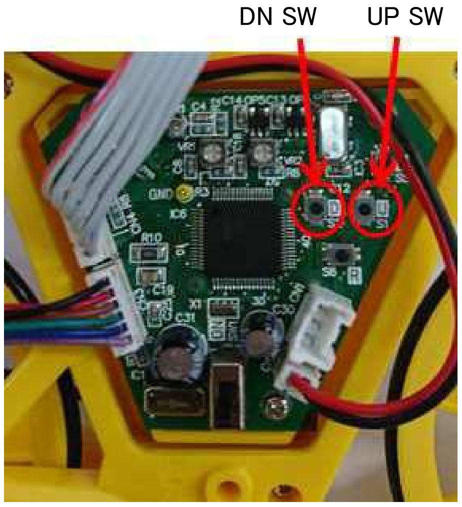
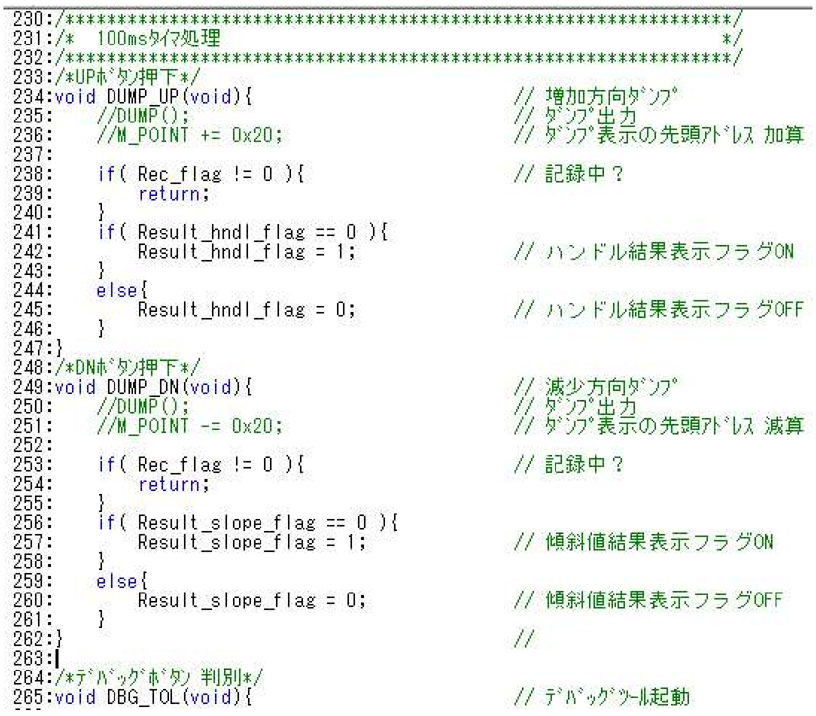


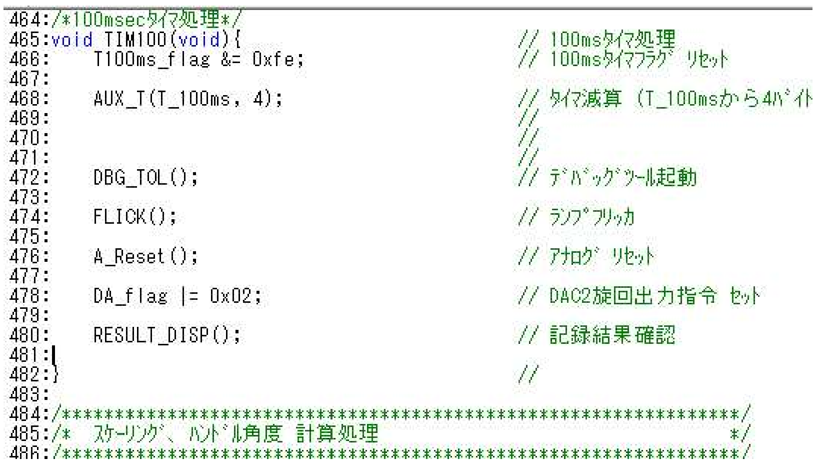
Photo 6.2 Body UP・ DN button SW



```
/*********************************************************************/
/*  100msﾀｲﾏ処理                                                     */
/*********************************************************************/
/*UPﾎﾞﾀﾝ押下*/
void DUMP_UP(void){                               // 増加方向ﾀﾞﾝﾌﾟ
    //DUMP();                                     // ﾀﾞﾝﾌﾟ出力
    //M_POINT += 0x20;                            // ﾀﾞﾝﾌﾟ表示の先頭ｱﾄﾞﾚｽ 加算

    if( Rec_flag != 0 ){                          // 記録中？
        return;
    }
    if( Result_hndl_flag == 0 ){
        Result_hndl_flag = 1;                     // ハンドル結果表示フラグON
    }
    else{
        Result_hndl_flag = 0;                     // ハンドル結果表示フラグOFF
    }
}
/*DNﾎﾞﾀﾝ押下*/
void DUMP_DN(void){                               // 減少方向ﾀﾞﾝﾌﾟ
    //DUMP();                                     // ﾀﾞﾝﾌﾟ出力
    //M_POINT -= 0x20;                            // ﾀﾞﾝﾌﾟ表示の先頭ｱﾄﾞﾚｽ 減算

    if( Rec_flag != 0 ){                          // 記録中？
        return;
    }
    if( Result_slope_flag == 0 ){
        Result_slope_flag = 1;                    // 傾斜値結果表示フラグON
    }
    else{
        Result_slope_flag = 0;                    // 傾斜値結果表示フラグOFF
    }
}                                                 //

/*ﾃﾞﾊﾞｯｸﾞﾎﾞﾀﾝ 判別*/
void DBG_TOL(void){                               // ﾃﾞﾊﾞｯｸﾞﾂｰﾙ起動
```

Image 6.20 Result display control

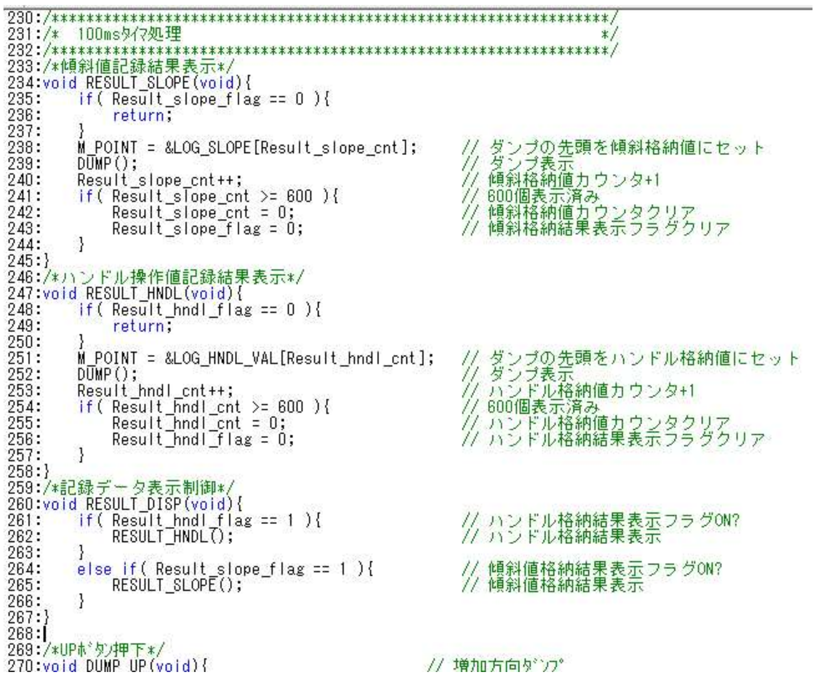
⑦ Write a call to the recording flag check function 'RESULT_DISP( )' in the function 'TIM 100( )' (Image 6.21). The function called in the function 'TIM100( )' is executed every 100 msec.



```
/*100msecﾀｲﾏ処理*/
void TIM100(void){                                  // 100msﾀｲﾏ処理
    T100ms_flag &= 0xfe;                            // 100msﾀｲﾏﾌﾗｸﾞ ﾘｾｯﾄ

    AUX_T(T_100ms, 4);                              // ﾀｲﾏ減算 (T_100msから4ﾊﾞｲﾄ
                                                    //
                                                    //
                                                    //
    DBG_TOL();                                      // ﾃﾞﾊﾞｯｸﾞﾂｰﾙ起動

    FLICK();                                        // ﾗﾝﾌﾟﾌﾘｯｶ

    A_Reset();                                      // ｱﾅﾛｸﾞ ﾘｾｯﾄ

    DA_flag |= 0x02;                                // DAC2旋回出力指令 ｾｯﾄ

    RESULT_DISP();                                  // 記録結果確認

}                                                   //

/******************************************************************/
/*  ｽｹｰﾘﾝｸﾞ、ﾊﾝﾄﾞﾙ角度 計算処理                                      */
/******************************************************************/
```

Image 6.21 100msec call

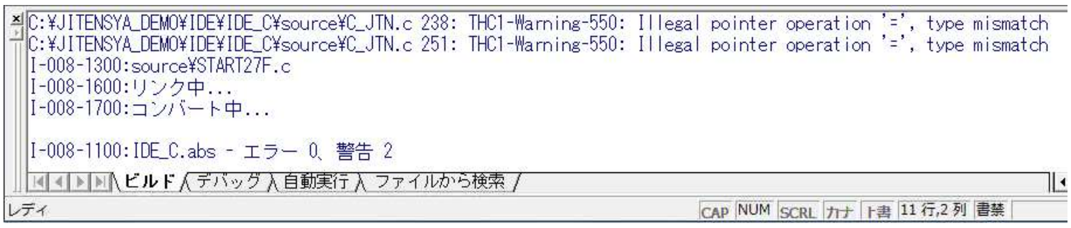
⑧ Write the functions 'RESULT_SLOPE( )', 'RESULT_HNDL( )', and 'RESULT_DISP( )' below the comment for the 100msec timer processing (Image 6. 22). The function 'RESULT_DISP( )' judges the flags set by the UP and DN buttons on the vehicle board every 100msec. At each call destination, the address of the array is stored in M_POINT. The function 'DUMP( )' immediately afterwards displays the stored results on the screen. After the results have been displayed 600 times, the flag is cleared and the display of the results is stopped.



```
/******************************************************************/
/*  100msﾀｲﾏ処理                                                   */
/******************************************************************/
/*傾斜値記録結果表示*/
void RESULT_SLOPE(void){
    if( Result_slope_flag == 0 ){
        return;
    }
    M_POINT = &LOG_SLOPE[Result_slope_cnt];     // ダンプの先頭を傾斜格納値にセット
    DUMP();                                     // ダンプ表示
    Result_slope_cnt++;                         // 傾斜格納値カウンタ+1
    if( Result_slope_cnt >= 600 ){              // 600個表示済み
        Result_slope_cnt = 0;                   // 傾斜格納値カウンタクリア
        Result_slope_flag = 0;                  // 傾斜格納結果表示フラグクリア
    }
}
/*ハンドル操作値記録結果表示*/
void RESULT_HNDL(void){
    if( Result_hndl_flag == 0 ){
        return;
    }
    M_POINT = &LOG_HNDL_VAL[Result_hndl_cnt];   // ダンプの先頭をハンドル格納値にセット
    DUMP();                                     // ダンプ表示
    Result_hndl_cnt++;                          // ハンドル格納値カウンタ+1
    if( Result_hndl_cnt >= 600 ){               // 600個表示済み
        Result_hndl_cnt = 0;                    // ハンドル格納値カウンタクリア
        Result_hndl_flag = 0;                   // ハンドル格納結果表示フラグクリア
    }
}
/*記録データ表示制御*/
void RESULT_DISP(void){
    if( Result_hndl_flag == 1 ){                // ハンドル格納結果表示フラグON?
        RESULT_HNDL();                          // ハンドル格納結果表示
    }
    else if( Result_slope_flag == 1 ){          // 傾斜値格納結果表示フラグON?
        RESULT_SLOPE();                         // 傾斜値格納結果表示
    }
}

/*UPﾎﾞﾀﾝ押下*/
void DUMP_UP(void){                           // 増加方向ﾀﾞﾝﾌﾟ
```

Image 6.22 Result display control

⑨After writing the above program, run the build and check that there are no errors. A warning message may appear, but this will not affect the operation (Image 6. 23).

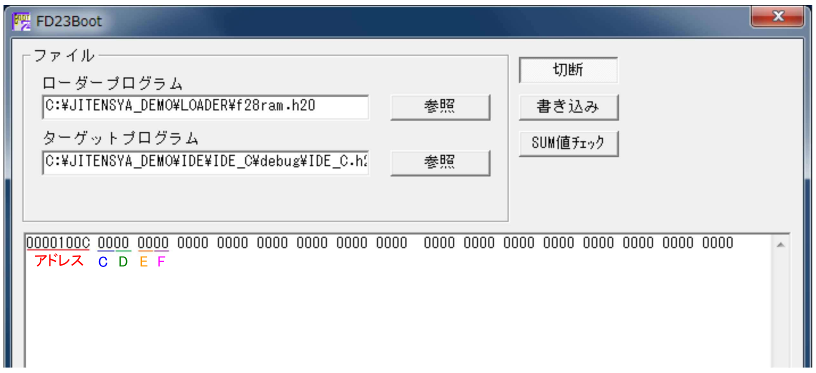


```
C:¥JITENSYA_DEMO¥IDE¥IDE_C¥source¥C_JTN.c 238: THC1-Warning-550: Illegal pointer operation '=', type mismatch
C:¥JITENSYA_DEMO¥IDE¥IDE_C¥source¥C_JTN.c 251: THC1-Warning-550: Illegal pointer operation '=', type mismatch
I-008-1300:source¥START27F.c
I-008-1600:リンク中...
I-008-1700:コンバート中...

I-008-1100:IDE_C.abs - エラー 0、警告 2
```

Image 6.23 Build result

⑩ If there are no errors, connect the vehicle to the debug tool, start FD23Boot. exe, and write the program. Use FD23Boot as is to check that it is working properly. After resetting the vehicle, press the UP button on the vehicle's board to display the recording results. Since no command to start recording was issued from the remote control, the displayed result is "00" in hexadecimal (Image 6 . 24).

In this case, the displayed results are read as follows: the first "100C" is the variable address (the address may not be "100C"; see ⑪).
The first "00" after that is the storage result of the first address "100C", followed by "100D", "100E", and so on. Also, the variable LOG_HNDLE_VAL[ ] displayed here is of type int, so it is expressed in 2 bytes. Therefore, it is read as the concatenated value of "100D" and "100C". 32 bytes are displayed per line, but to take into account the work of creating a graph, the next line is displayed starting from the address 2 bytes away.

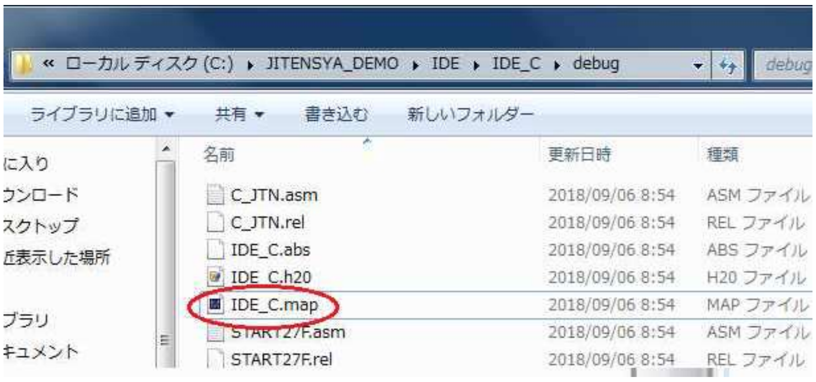


FD23Boot
ファイル
ローダープログラム: C:¥JITENSYA_DEMO¥LOADER¥f28ram.h20
ターゲットプログラム: C:¥JITENSYA_DEMO¥IDE¥IDE_C¥debug¥IDE_C.h

```
0000100C 0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000
```

Image 6.24 How to read the results

⑪Check the starting address of the variable LOG_HNDLE_VAL[ ]. Open the map file "IDE_C.map" output by the compiler. If you are using an IDE environment, the map file is located in the debug folder in the project folder (Image 6.25). Search for LOG_HNDLE_VAL in the map file and check the address. In Image 6.26, you can see that the address is "100C". Also, the starting address of LOG_SLOPE is "14BC". Press the DN button switch on the vehicle board to check that debugging starts from "14BC".


« ローカル ディスク (C:) ▸ JITENSYA_DEMO ▸ IDE ▸ IDE_C ▸ debug

名前	更新日時	種類
C_JTN.asm	2018/09/06 8:54	ASM ファイル
C_JTN.rel	2018/09/06 8:54	REL ファイル
IDE_C.abs	2018/09/06 8:54	ABS ファイル
IDE_C.h20	2018/09/06 8:54	H20 ファイル
IDE_C.map	2018/09/06 8:54	MAP ファイル
START27F.asm	2018/09/06 8:54	ASM ファイル
START27F.rel	2018/09/06 8:54	REL ファイル

画像6.25　マップファイル

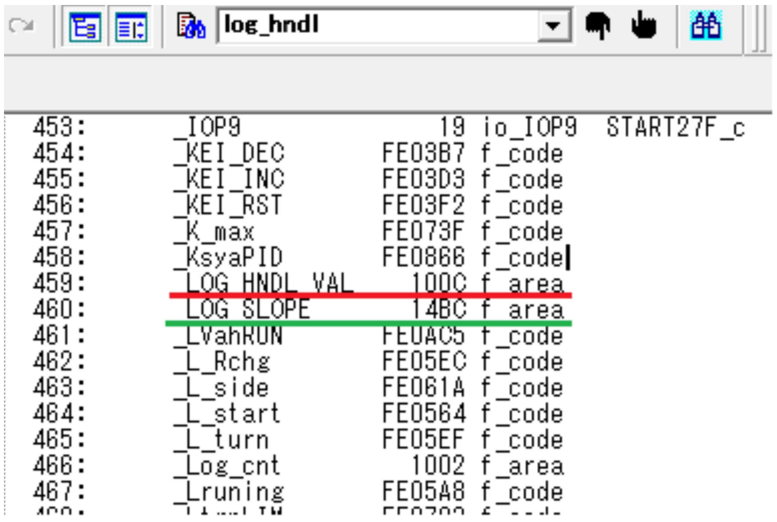
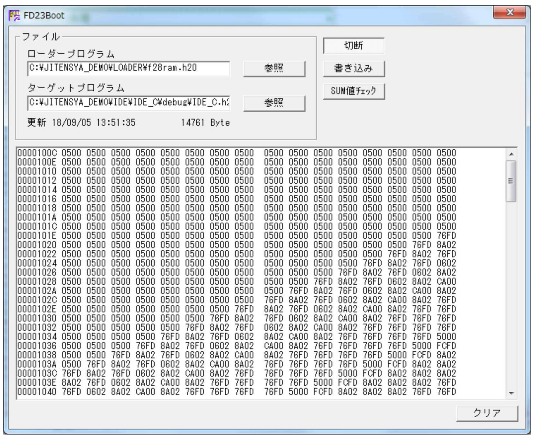

```
453:    _IOP9              19 io_IOP9  START27F_c
454:    _KEI_DEC       FE03B7 f_code
455:    _KEI_INC       FE03D3 f_code
456:    _KEI_RST       FE03F2 f_code
457:    _K_max         FE073F f_code
458:    _KsyaPID       FE0866 f_code
459:    _LOG_HNDL_VAL    100C f_area
460:    _LOG_SLOPE       14BC f_area
461:    _LVahRUN       FE0AC5 f_code
462:    _L_Rchg        FE05EC f_code
463:    _L_side        FE061A f_code
464:    _L_start       FE0564 f_code
465:    _L_turn        FE05EF f_code
466:    _Log_cnt         1002 f_area
467:    _Lruning       FE05A8 f_code
```

Image 6.26 Address verification

⑫ Once you have confirmed that the displayed addresses and variables are correct, reset the vehicle and check that the values have been stored. Press the UP button on the remote control to start recording, and operate the handlebar volume for a while. After a certain period of time, press the DOWN button on the remote control to stop recording. At this time, make sure that the LED lamp goes out. Press the UP button on the vehicle to display the recording results and confirm that the value has changed (Image 6.27).

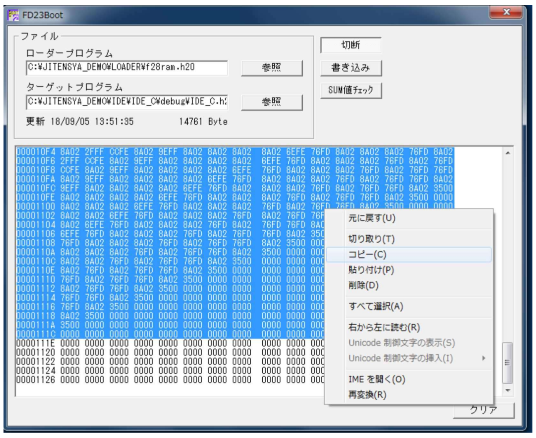
⑬Edit the data to confirm the displayed values as numbers. This time, we are using Microsoft Excel 2010. Select the recorded value from the top and copy it (Image 6 .28). Open Excel and paste the copied value. Select Delimiter from the Data tab to launch the Delimiter Wizard. Set it to separate by space (Image 6.29). Select Next, and in the Select Data Format for the Separated Column, select the first two in the data preview and set it to text format. Select Finish to check the data (Image 6 .30).
Erase the data after separation, leaving only the address cell column and the first data cell column.



C:¥JITENSYA_DEMO¥LOADER¥f28ram.h20
C:¥JITENSYA_DEMO¥IDE¥IDE_C¥debug¥IDE_C.h
更新 18/09/05 13:51:35 14761 Byte

```
0000100C 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
0000100E 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
00001010 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
00001012 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
00001014 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
00001016 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
00001018 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
0000101A 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
0000101C 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 0500
0000101E 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 0500 76FD
00001020 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 0500 76FD 8A02
00001022 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500 76FD 8A02 76FD
00001024 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 76FD 8A02 76FD 0602
00001026 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 76FD 8A02 76FD 0602 8A02
00001028 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 76FD 8A02 76FD 0602 8A02 CA00
0000102A 0500 0500 0500 0500 0500 0500 0500 0500  0500 76FD 8A02 76FD 0602 8A02 CA00 8A02
0000102C 0500 0500 0500 0500 0500 0500 0500 0500  76FD 8A02 76FD 0602 8A02 CA00 8A02 76FD
0000102E 0500 0500 0500 0500 0500 0500 0500 76FD  8A02 76FD 0602 8A02 CA00 8A02 76FD 76FD
00001030 0500 0500 0500 0500 0500 0500 76FD 8A02  76FD 0602 8A02 CA00 8A02 76FD 76FD 76FD
00001032 0500 0500 0500 0500 0500 76FD 8A02 76FD  0602 8A02 CA00 8A02 76FD 76FD 76FD 76FD
00001034 0500 0500 0500 0500 76FD 8A02 76FD 0602  8A02 CA00 8A02 76FD 76FD 76FD 76FD 5000
00001036 0500 0500 0500 76FD 8A02 76FD 0602 8A02  CA00 8A02 76FD 76FD 76FD 76FD 5000 FCFD
00001038 0500 0500 76FD 8A02 76FD 0602 8A02 CA00  8A02 76FD 76FD 76FD 76FD 5000 FCFD 8A02
0000103A 0500 76FD 8A02 76FD 0602 8A02 CA00 8A02  76FD 76FD 76FD 76FD 5000 FCFD 8A02 8A02
0000103C 76FD 8A02 76FD 0602 8A02 CA00 8A02 76FD  76FD 76FD 76FD 5000 FCFD 8A02 8A02 8A02
0000103E 8A02 76FD 0602 8A02 CA00 8A02 76FD 76FD  76FD 76FD 5000 FCFD 8A02 8A02 8A02 76FD
00001040 76FD 0602 8A02 CA00 8A02 76FD 76FD 76FD  76FD 5000 FCFD 8A02 8A02 8A02 76FD 76FD
```

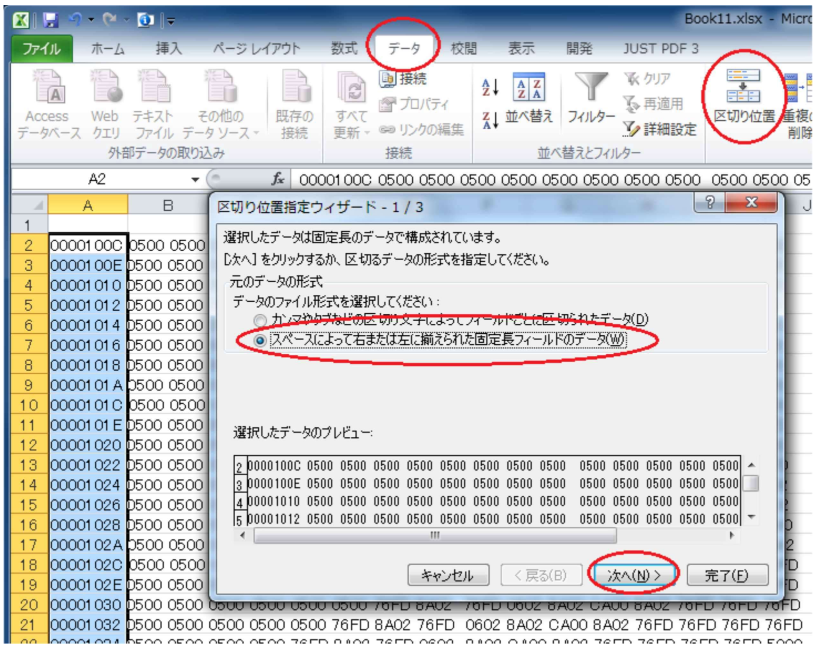
Image 6.27 Checking the recorded contents



C:¥JITENSYA_DEMO¥LOADER¥f28ram.h20
C:¥JITENSYA_DEMO¥IDE¥IDE_C¥debug¥IDE_C.h
更新 18/09/05 13:51:35 14761 Byte

```
000010F4 8A02 2FFF CCFE 8A02 9EFF 8A02 8A02 8A02  8A02 6EFE 76FD 8A02 8A02 8A02 76FD 8A02
000010F6 2FFF CCFE 8A02 9EFF 8A02 8A02 8A02 8A02  6EFE 76FD 8A02 8A02 8A02 76FD 8A02 76FD
000010F8 CCFE 8A02 9EFF 8A02 8A02 8A02 8A02 6EFE  76FD 8A02 8A02 8A02 76FD 8A02 76FD 76FD
000010FA 8A02 9EFF 8A02 8A02 8A02 8A02 6EFE 76FD  8A02 8A02 8A02 76FD 8A02 76FD 76FD 8A02
000010FC 9EFF 8A02 8A02 8A02 8A02 6EFE 76FD 8A02  8A02 8A02 76FD 8A02 76FD 76FD 8A02 3500
000010FE 8A02 8A02 8A02 8A02 6EFE 76FD 8A02 8A02  8A02 76FD 8A02 76FD 76FD 8A02 3500 0000
00001100 8A02 8A02 8A02 6EFE 76FD 8A02 8A02 8A02  76FD 8A02 76FD 76FD 8A02 3500 0000 0000
00001102 8A02 8A02 6EFE 76FD 8A02 8A02 8A02 76FD  8A02 76FD 76F
00001104 8A02 6EFE 76FD 8A02 8A02 8A02 76FD 8A02  76FD 76FD 8A0
00001106 6EFE 76FD 8A02 8A02 8A02 76FD 8A02 76FD  76FD 8A02 350
00001108 76FD 8A02 8A02 8A02 76FD 8A02 76FD 76FD  8A02 3500 000
0000110A 8A02 8A02 8A02 76FD 8A02 76FD 76FD 8A02  3500 0000 000
0000110C 8A02 8A02 76FD 8A02 76FD 76FD 8A02 3500  0000 0000 000
0000110E 8A02 76FD 8A02 76FD 76FD 8A02 3500 0000  0000 0000 000
00001110 76FD 8A02 76FD 76FD 8A02 3500 0000 0000  0000 0000 000
00001112 8A02 76FD 76FD 8A02 3500 0000 0000 0000  0000 0000 000
00001114 76FD 76FD 8A02 3500 0000 0000 0000 0000  0000 0000 000
00001116 76FD 8A02 3500 0000 0000 0000 0000 0000  0000 0000 000
00001118 8A02 3500 0000 0000 0000 0000 0000 0000  0000 0000 000
0000111A 3500 0000 0000 0000 0000 0000 0000 0000  0000 0000 000
0000111C 0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 000
0000111E 0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 000
00001120 0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 000
00001122 0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 000
00001124 0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 000
00001126 0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 000
```

Image 6.28 Copy of recorded values



f_x 00001 00C 0500 0500 0500 0500 0500 0500 0500 0500 0500 0500 05

	A	B
1		
2	00001 00C	0500 0500
3	00001 00E	0500 0500
4	00001 010	0500 0500
5	00001 012	0500 0500
6	00001 014	0500 0500
7	00001 016	0500 0500
8	00001 018	0500 0500
9	00001 01 A	0500 0500
10	00001 01 C	0500 0500
11	00001 01 E	0500 0500
12	00001 020	0500 0500
13	00001 022	0500 0500
14	00001 024	0500 0500
15	00001 026	0500 0500
16	00001 028	0500 0500
17	00001 02A	0500 0500
18	00001 02C	0500 0500
19	00001 02E	0500 0500
20	00001 030	0500 0500 0500 0500 0500 0500 76FD 8A02 76FD 0602 8A02 CA00 8A02 76FD 76FD 76FD
21	00001 032	0500 0500 0500 0500 0500 76FD 8A02 76FD 0602 8A02 CA00 8A02 76FD 76FD 76FD 76FD

区切り位置指定ウィザード - 1 / 3

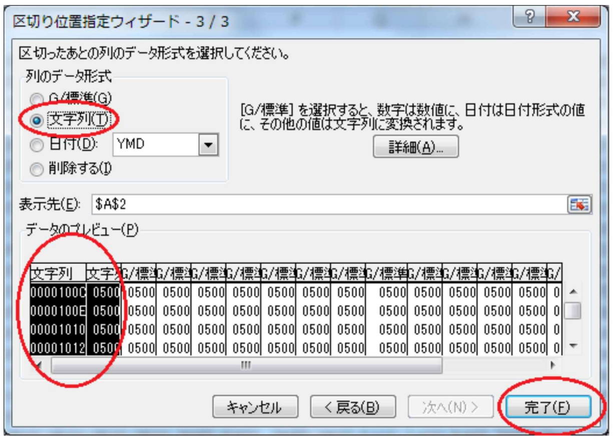
選択したデータは固定長のデータで構成されています。
[次へ] をクリックするか、区切るデータの形式を指定してください。

元のデータの形式
データのファイル形式を選択してください：
- ○ カンマやタブなどの区切り文字によってフィールドごとに区切られたデータ(D)
- [✓ ◉ スペースによって右または左に揃えられた固定長フィールドのデータ(W)]

選択したデータのプレビュー:

```
2 0000100C 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500
3 0000100E 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500
4 00001010 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500
5 00001012 0500 0500 0500 0500 0500 0500 0500 0500  0500 0500 0500 0500 0500
```

Image 6.29 Paste Excel



区切り位置指定ウィザード - 3 / 3

区切ったあとの列のデータ形式を選択してください。

列のデータ形式
- ○ G/標準(G)
- [✓ ◉ 文字列(T)]
- ○ 日付(D): YMD
- ○ 削除する(I)

[G/標準] を選択すると、数字は数値に、日付は日付形式の値に、その他の値は文字列に変換されます。

表示先(E): A2

データのプレビュー(P)

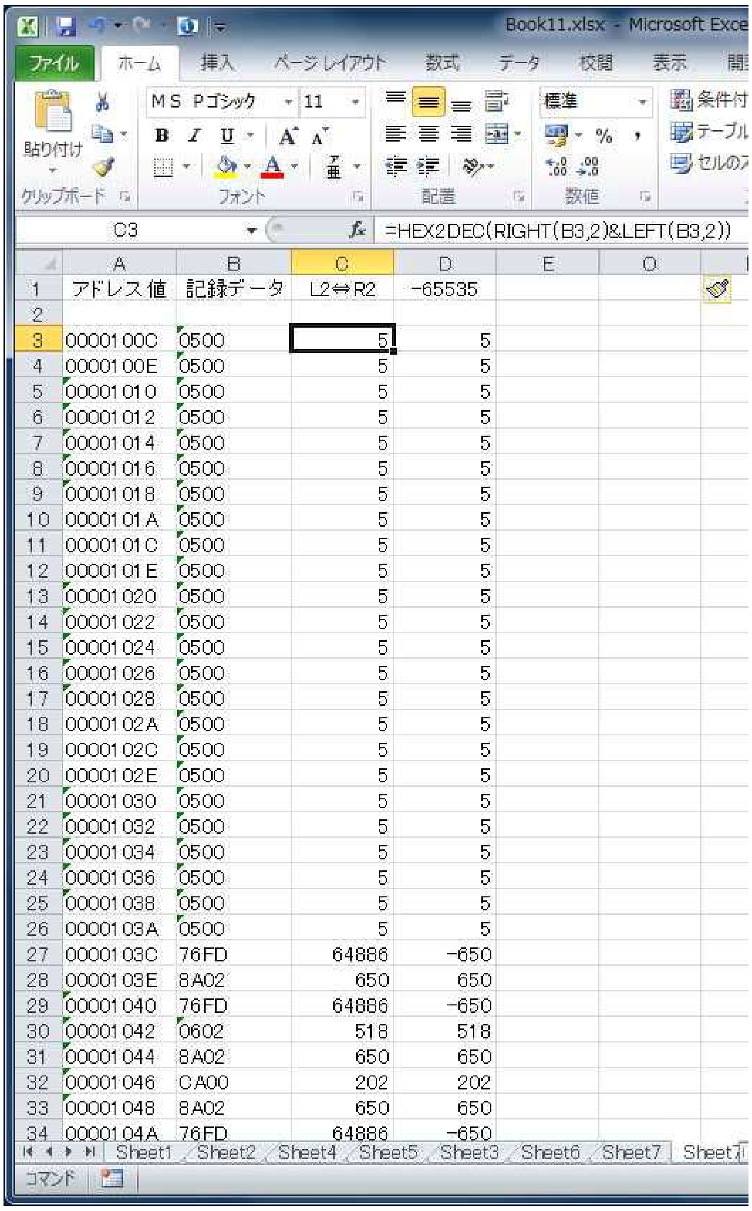
文字列	文字列	G/標準	G/標準	G/標準	G/標準	G/標準	G/標準	G/標準	G/標準	G/標準	G/標準	G/標準	G/
0000100C	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0
0000100E	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0
00001010	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0
00001012	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0500	0

Image 6.30 Delimiter setting

As shown in image 6.31, convert the recorded data in column B to a signed decimal number.
First, swap the two bytes before and after the recorded data. Convert the swapped result from hexadecimal to decimal and display the result in column C. The data converted to decimal is of signed int type, so if the value exceeds 3 2767, the maximum value on the positive side, subtract 65536 to convert it to
a signed decimal number. The converted result is in column D.

- Input formula for cell C3: =HEX2DEC(RIGHT(B3,2)&LEFT(B3,2))
- Input formula for cell D3: =IF(C3>=32767,C3-65536,C3)

You can confirm that column D is a signed decimal number from the address value "103C".
LOG_SLOPE is also of signed int type, so you can handle it in the same way.
After confirming the operation, turn off the power to the vehicle and remove the debug tool.



C3 =HEX2DEC(RIGHT(B3,2)&LEFT(B3,2))

	A	B	C	D	E	O
1	アドレス値	記録データ	L2⇔R2	-65535		
2						
3	0000100C	0500	5	5		
4	0000100E	0500	5	5		
5	00001010	0500	5	5		
6	00001012	0500	5	5		
7	00001014	0500	5	5		
8	00001016	0500	5	5		
9	00001018	0500	5	5		
10	0000101A	0500	5	5		
11	0000101C	0500	5	5		
12	0000101E	0500	5	5		
13	00001020	0500	5	5		
14	00001022	0500	5	5		
15	00001024	0500	5	5		
16	00001026	0500	5	5		
17	00001028	0500	5	5		
18	0000102A	0500	5	5		
19	0000102C	0500	5	5		
20	0000102E	0500	5	5		
21	00001030	0500	5	5		
22	00001032	0500	5	5		
23	00001034	0500	5	5		
24	00001036	0500	5	5		
25	00001038	0500	5	5		
26	0000103A	0500	5	5		
27	0000103C	76FD	64886	-650		
28	0000103E	8A02	650	650		
29	00001040	76FD	64886	-650		
30	00001042	0602	518	518		
31	00001044	8A02	650	650		
32	00001046	CA00	202	202		
33	00001048	8A02	650	650		
34	0000104A	76FD	64886	-650		

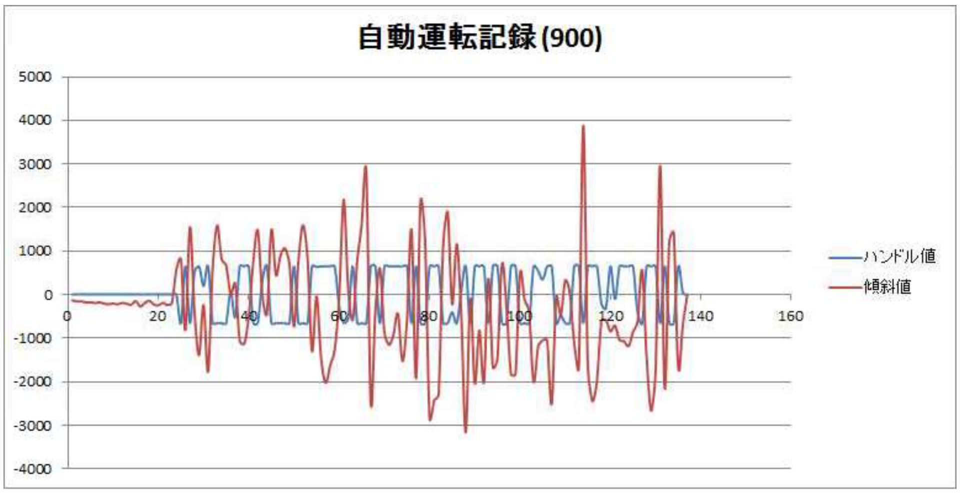
Image 6.31 Data conversion

⑭ Acquire data during automatic driving. Refer to the driving instruction video and perform automatic driving. Press the remote control UP button switch just before driving to ensure reliable data recording.

⑮After automatic operation has stopped, press the DOWN button on the remote control to stop recording. Please note that if you turn off the power to the vehicle, the recorded values will be erased.

⑯Connect the debug tool to the vehicle and start FD23Boot. Press the UP button on the vehicle board to display the recorded data in LOG_HNDLE_VAL. Refer to ⑬ to prepare a signed decimal number. Once the handlebar value has been converted, clear the display on FD23Boot. Press the DN button on the vehicle board to display the recorded data in LOG_SLOPE, and convert it to signed decimal data in the sameway.

⑰You can create a graph using the prepared signed decimal data (Graph 6.2). This graph shows data for approximately 100msec intervals, but you can change the program to change it to every 10msec, or add turning values, pedal speed, etc. to the data you obtain.



Graph 6.2 Driving record (900)

Chapter 7 Bicycle body manufacturing

In commercializing this automatic attitude control bicycle, we made many prototypes.

We have summarized the problems and information we learned in the process.

We hope this will be helpful when you build your own bicycle. Photos 7.2, 7.3, and 7.4 are some of the prototypes.

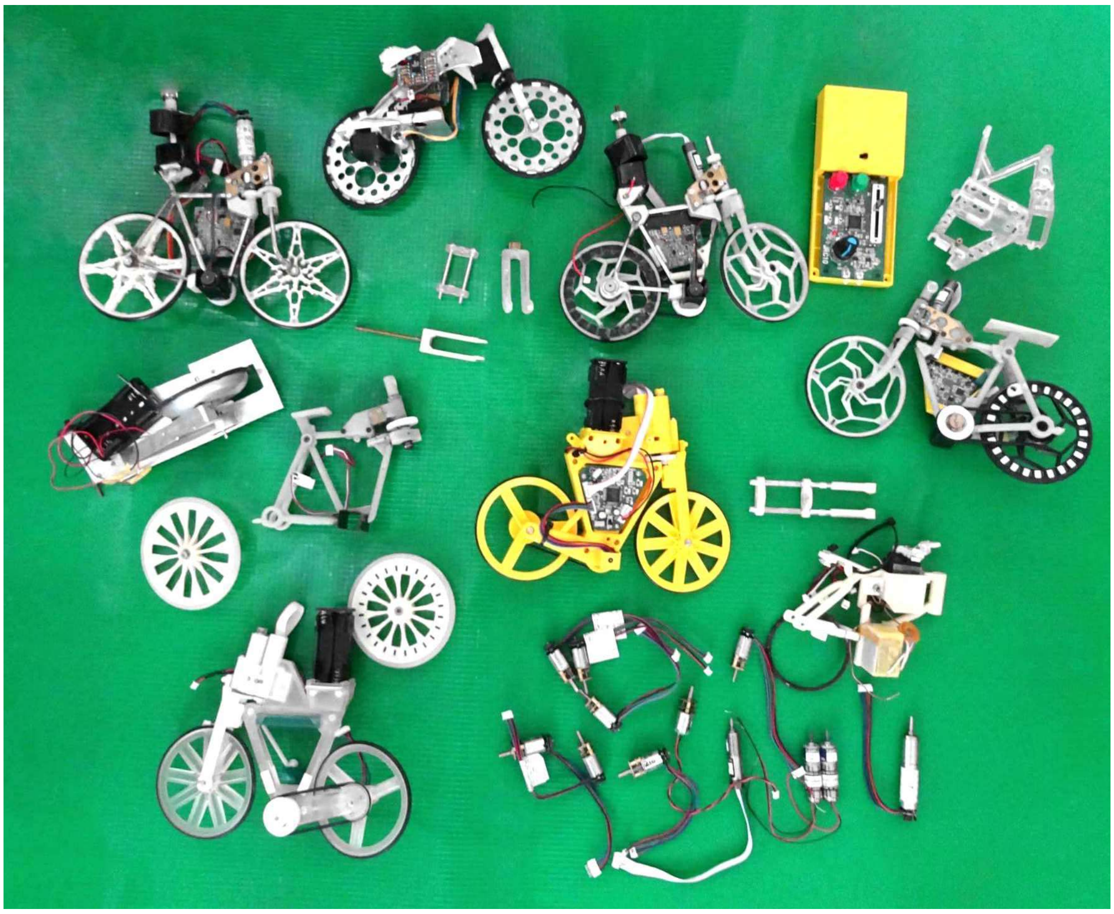


Photo 7.1

The commercialized bicycle, the prototype, and each part. In the lower right corner, you can see various geared motors with different reduction ratios.

Photo 7.2

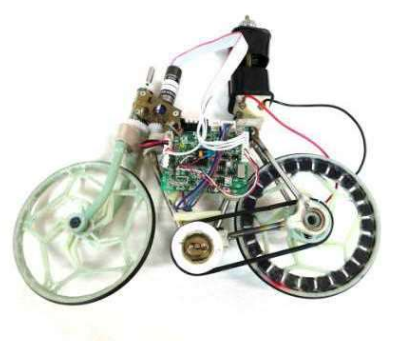


Photo 7.3

Photo 7.4

7.1 What is an automatic attitude control bicycle?

(1) Hardware

Wheel diameter	70mm	Approximately 1/10 of the actual size
Wheelbase	115mm	
Ground height	160mm	

(2) Control

CPU used: TMP91FW27UG (Toshiba)
: TMPM332FWUG (Toshiba)

Input signal: 1 Angular velocity sensor Tilt signal, rotation signal
2 Encoder signals Steering wheel angle, pedal speed

External input signal (infrared remote control): direction, speed, start trigger
Control output: Steering wheel operation motor drive, pedal drive motor drive

(3) Control method

How does it feel and move? There are three ways to maintain balance without falling over. The first is to move the fulcrum. The fulcrum must always be under the center of gravity. The second is to move the center of gravity itself. In fact, anyone can wiggle their body while the bicycle is stopped and not fall over for a while. The human being itself is measuring the movement of the center of gravity. The third is to move both the center of gravity and the fulcrum. This time, we did not use a gyro or centrifugal force effect, but only controlled the movement of the fulcrum.

7.2 Center of gravity and fulcrum

No matter what shape an object has, there is always a point called the center of gravity. If it is solid, the center of gravity is in a fixed position, and if it moves like a human, it has a fixed center of gravity at every moment of its pose. If you connect that point with a string and hang the object from it, it will be stable without any lopsided movement. It will continue to maintain the same state. This is called a statically stable state. Now, if you hang a weight on the string, the string will point vertically. Usually, the center of gravity is inside the object, so it is not possible to actually connect a string to that point, but if you bring the fulcrum to the line that points vertically, the object will be stable.

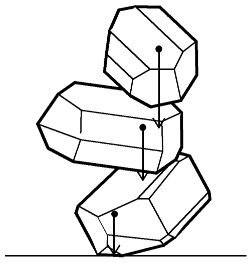


Figure 7.1

As shown in Figure 7.1, you may have seen a video of a riverbank where stones are precariously stacked one by one to balance, but in this case the vertical line of the center of gravity of the top stone passes through the tangent point with the second stone, and the vertical line of the center of gravity of the first and second stones together passes through the tangent point with the third stone . (The tangent point is the fulcrum.)

Furthermore, the center of gravity of the human body when standing with both arms down is located almost behind the navel, and if the point where the vertical line of the center of gravity meets the ground is within the area where both feet touch the ground, the body will be stable without wobbling, but if it is outside this area , the body will lose balance and fall (Figure 7.2).

In other words, when you lift your right foot to step forward, your center of gravity is in front of your body, and if you continue like this you will fall forward. However, you can stabilize yourself by finding a fulcrum below that vertical line. Specifically, you can achieve stability by landing on your right foot and making sure that a vertical line intersects with the line connecting the two points where your right and left feet touch the ground. By always keeping the fulcrum below your moving center of gravity, you can maintain your balance.

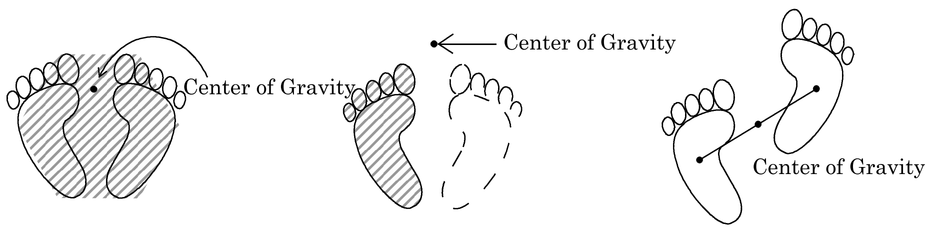
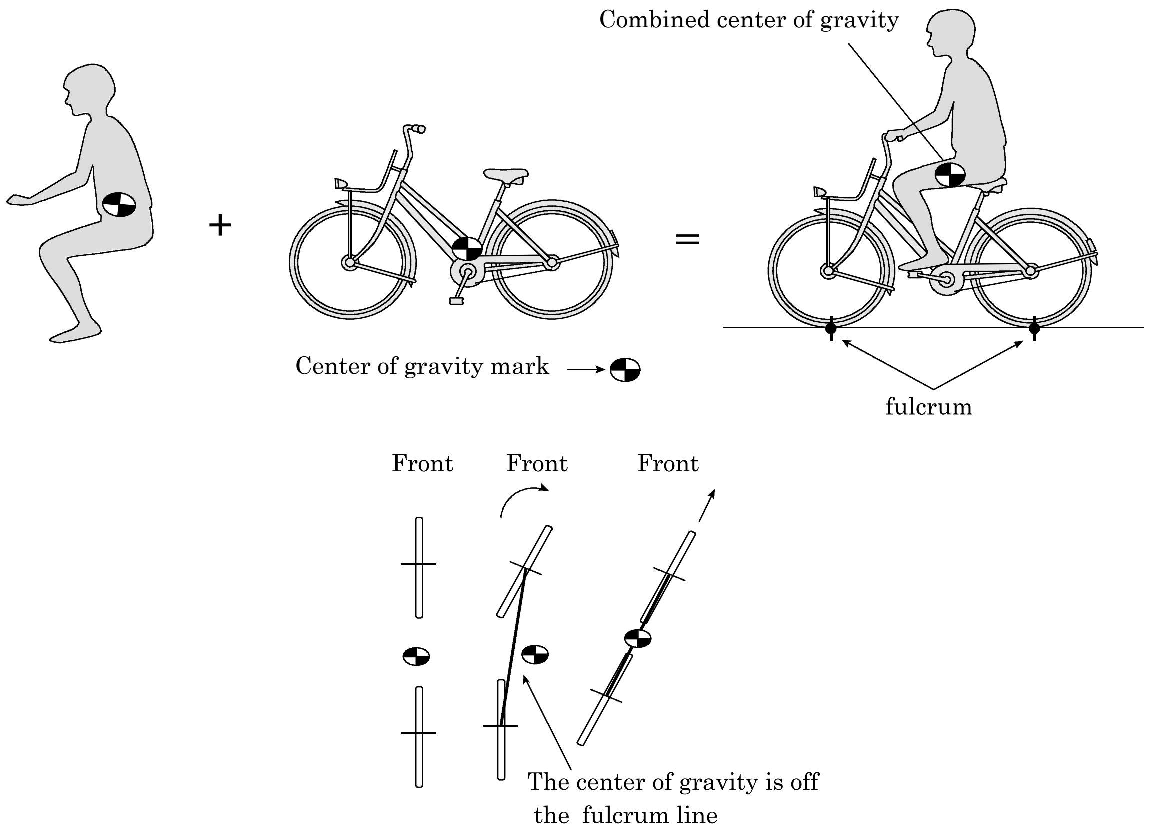


Figure 7.2

(1) In the case of bicycles
What does it mean for a bicycle to fall over? It means that the center of gravity of the combined object of the human and the bicycle moves away from the line (fulcrum) connecting the two points where the front and rear wheels touch the ground, causing the bicycle to lose balance.

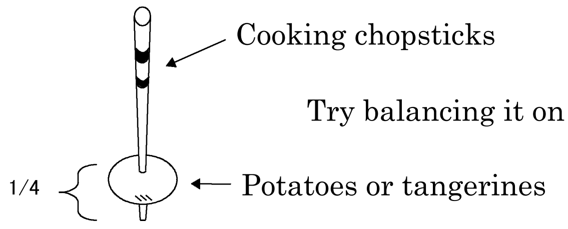

Combined center of gravity
+
=
Center of gravity mark
fulcrum
Front
Front
Front
The center of gravity is off the fulcrum line
Figure 7.3

To keep a bicycle from falling over, you only need to consider left -right balance, not front-to-back balance like with a unicycle. (Figure 7.4, left) If the bicycle leans to the left or right, balance can be maintained by constantly keeping the fulcrum under the bicycle's shifting center of gravity. With a bicycle, the fulcrum is not a single point, but a support line connecting the two wheels. (Figure 7.3, center)

(2) Moving the fulcrum

So how do we move the fulcrum (support line)? First, when the tilt sensor detects the direction (right or left) and the degree of tilt, it quickly turns the handlebars in the direction of the tilt. Of course, just turning the handlebars is not enough; you can move the support line by stepping on the pedals and moving the bicycle forward. And it has to get closer to the support line faster than the speed of the tilt (the speed at which the center of gravity moves away from the support line). In the same way, the support line will move quickly to balance out the next change in tilt.

Now, let's do an experiment. Using items you have in your kitchen, make the following:



Try balancing it on the palm of your hand.

I think it's possible to do it somehow, but it will be easier if you invert it like A. This is because it will be easier to adjust the fulcrum to the movement of the center of gravity than if the potato was on the bottom.

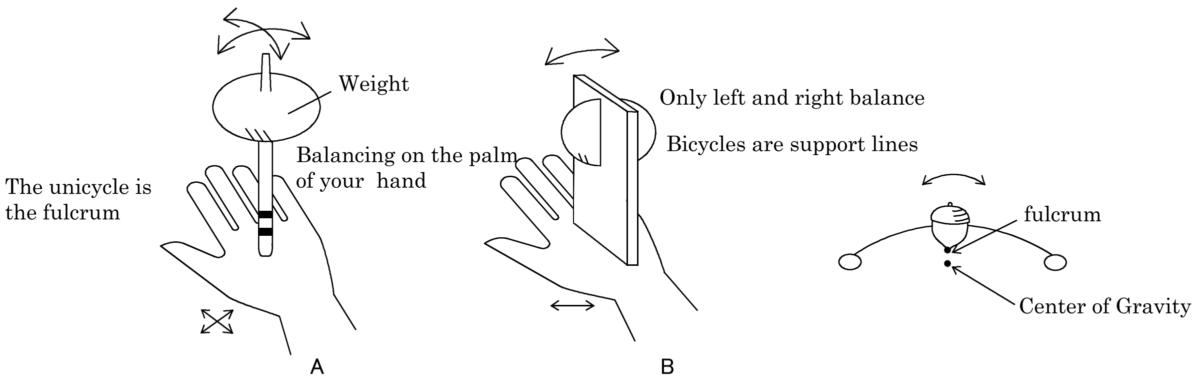

Weight
Balancing on the palm of your hand
The unicycle is the fulcrum
A
Only left and right balance
Bicycles are support lines
B
fulcrum
Center of Gravity

Figure 7.4

By the way, there is a toy called a balancing toy (Figure 7.4, right), whose center of gravity is outside the body and whose fulcrum is above the center of gravity when in normal use. Therefore, even if an external force is applied to it, it will not fall over, but will sway back and forth and converge.

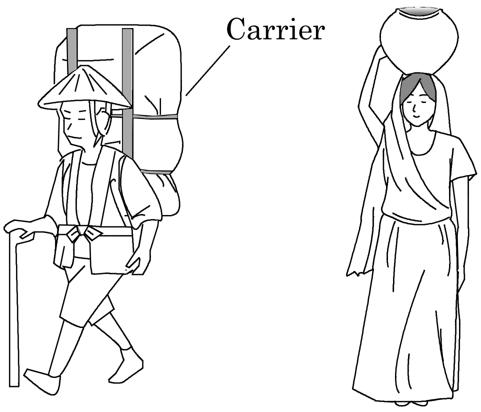


Figure 7.5

There is a celebrity named Ayame Goriki, and there is also someone with the same name and profession.

In the past, the job of these strong men (also called "powerful") was to carry the daily necessities of ascetic monks while running through the mountains and fields together with them. Nowadays, they carry supplies to mountain huts and act as guides on mountain climbs, carrying heavy equipment carried by climbers.

The amount of luggage they carry at one time is usually several dozen kilograms. Sometimes it can be just under 100 kilograms. And the way they carry it is by stacking the boxes high on their backs so that most of the luggage is above their heads. (This places their center of gravity farther away from the ground.) This high center of gravity makes it easier to move sideways.

If we were to carry the same weight of luggage, distributed across our waists, sideways movement (walking) would be heavy and difficult, even if the weight of each foot on the ground was the same . This is a kind of human wisdom; one example of this is the ancient practice of carrying a heavy pot of water on one's head (Figure 7.5).

7.3 Scale Ratio and Precision

The scale of an agile bicycle is 1/10 of the real thing. If the tolerance of a certain mechanism on a real bicycle is 3mm, the tolerance of an agile bicycle is 0.3mm.

Unless the absolute value is within 0.3 mm, precision is not achieved. Errors and play that are no problem in the 1:1 world are not acceptable in the 1/10 world.

In fact, we had a hard time controlling the handling of the bicycle. If the geared motor used here has any backlash or play, it will respond slowly and will tip over. We tried various gear ratios and types, and finally decided on one with an output shaft radial backlash of 0.07 mm or less (Figure 7.6) (By the way, the backlash of a pedal-driven geared motor is 0.5 mm.) This is a precision type that uses a coreless motor and responds quickly to reversible rotation signals. (It is the most expensive of the parts used, so it's a bit of a shame to use it.)

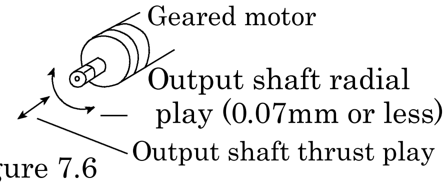


Figure 7.6

7.4 Rigidity and weight reduction

It is said that the lighter something is, the better it performs when it floats in the air (airplanes) or runs on the ground (cars). In the case of bicycles, they need to be light to improve maneuverability and to respond quickly to the movement of the support lines. On the other hand, the front wheel arm support (using two bearings) and the handle drive geared motor transmit power via spur gears.

If the frame that holds them in place is soft and easily twisted, torque will not be transmitted 100%.The force of transmission will be lost and reaction time will be delayed. This part requires strength rather than lightness.
Furthermore, this mechanism requires high precision in terms of gear fits, etc.

7.5 Components

(1) Batteries: Four AAA batteries make the bike's heaviest part. The center of gravity is raised by placing it in the same position as the heaviest person on a real bicycle As in the example in the previous section, heavy objects are elevated to make lateral movement easier.

(2)Handle drive geared motor (with encoder)

This is a key control part, and since it rotates reversibly, we need to consider an installation method that will prevent backlash. Initially, we used a servo used in hobby radio-controlled cars (Photo 7.2). The reason for this is that: 1) the unit has a built-in gear reduction device. 2) there is a potentiometer directly connected to the output shaft inside, which can be used as feedback for control. (The existing amplifier is not used.) 3) a gear can be machined and attached to the output shaft, allowing the handle shaft to be driven via the gear. 4) a bearing is inserted in the output shaft, so there is little backlash. When building such a prototype, using a radio-controlled servo is one practical option. Toy manufacturers also use radio-controlled servos in their prototyping.

(3) Pedal-driven geared motor (with encoder)

This is also an important part and is used for one-way rotation. Similarly, a servo was used for this part as well. The reasons are: 1) It is a gear reduction device as a unit. 2) A pulley can be attached to the output shaft and the rear wheel can be driven with an O-ring belt. Alternatively, the rear wheel can be attached directly to the output shaft. 3) A bearing is inserted in the output shaft, so there is little backlash.

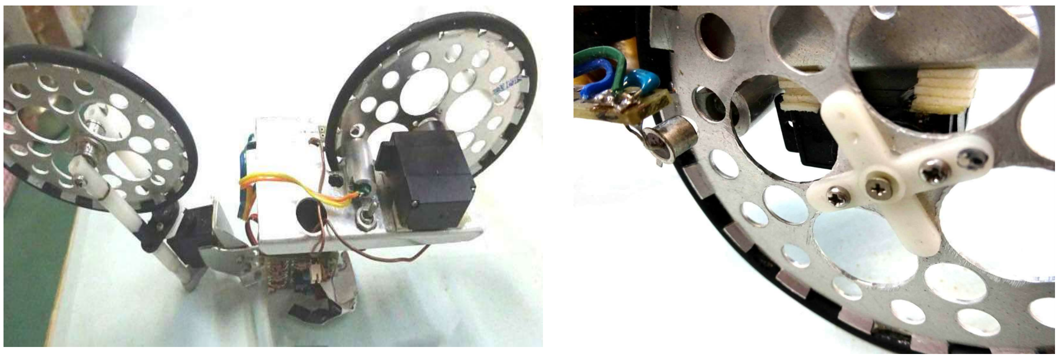


Photo 7.5
An example of a direct drive shaft with a rear wheel attached to the servo shaft. The case is simply fixed to the chassis with a sponge cushion. The photo on the right shows the light-emitting and light-receiving elements for the pulse encoder.

I removed the potentiometer and cut off the stopper on the output shaft gear so that it would rotate in one direction.

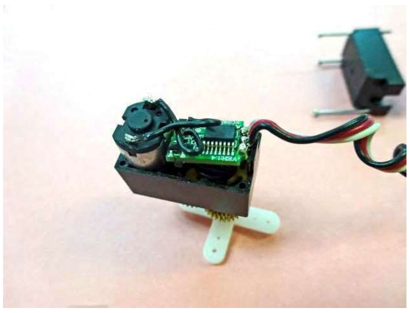


Photo 7.6
Removing the bottom of the servo case reveals the motor and amplifier board.

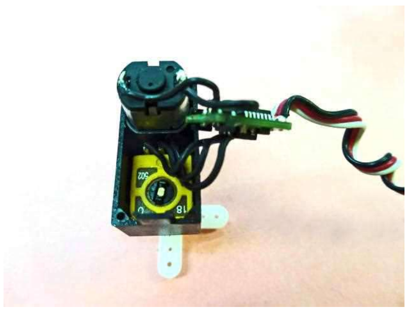


Photo 7.7
You can see the potentiometer under the amplifier board.

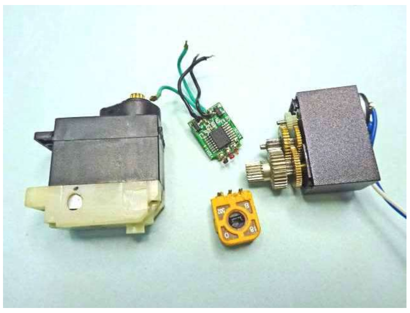


Photo 7.8
The amplifier board and potentiometer taken out.

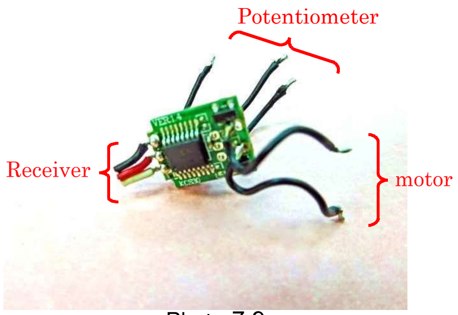


Photo 7.9
The servo amplifier board has three leads coming from the receiver, three going to the potentiometer, and two going to the motor.

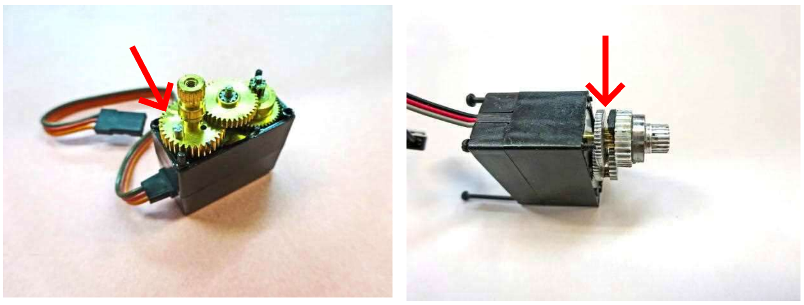


Photo 7.10
This is a stopper that limits the operating angle of the servo's main shaft gear.

The rotation feedback for the pedal drive was achieved by using the transmission pulse of a photodiode through the rear wheel. In this way, by modifying the servo, it was possible to create a bicycle that could run without falling over, but when it came to mass production, each modification had to be done manually, so it was not possible to do so. This is very useful when making your own prototype. Both the handle motor and the pedal motor are power sources, so they become a source of vibration for the body, and these vibrations become a noise source for the angular velocity sensor, so various ingenuity is required in the selection and installation of parts, etc. When manufacturing a bicycle, countermeasures against vibration are one of the most important issues.

(4) Frame

Molded resin has moderate rigidity and is lightweight, making it suitable for mass production. For prototyping, various materials are used.

・ Thin (2, 3 mm) wooden board

Easy to process. Screws or wood screws are used to attach parts.

・ Aluminum plate (0.5 to 2 mm)

Easy to bend and drill holes. Also used for wheels.

・ Stainless steel pipe (inner diameter ϕ2.1, outer diameter ϕ2.5 / inner diameter ϕ3.0 , outer diameter ϕ4.0)

Cut and use for front forks and tubes.

・ Brass plate (0.25, 0.8 mm: used for shims)

Used in areas that require precision and strength. Joined with silver solder. Silver solder can also be used to join other metals besides aluminum.

・ ABS resin plate (plastic plate 1.0, 1.5, 2.0 mm thick)

Easy to cut and bend, and uses a special adhesive.

(5) Control Board

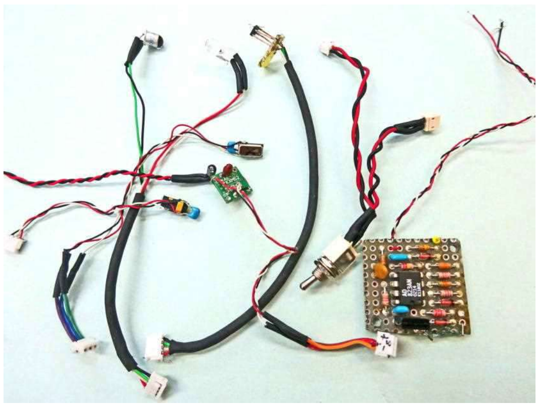


Photo 7.11

We also have a variety of leaded light-emitting and light-receiving elements, as well as infrared receiving amplifier boards.

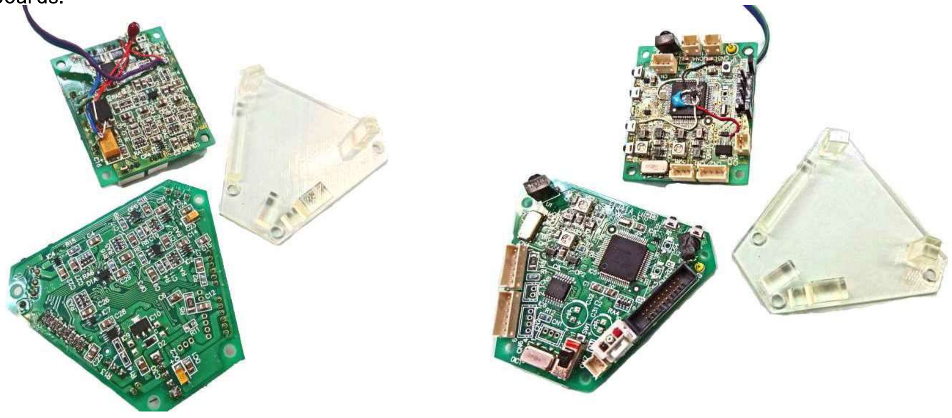
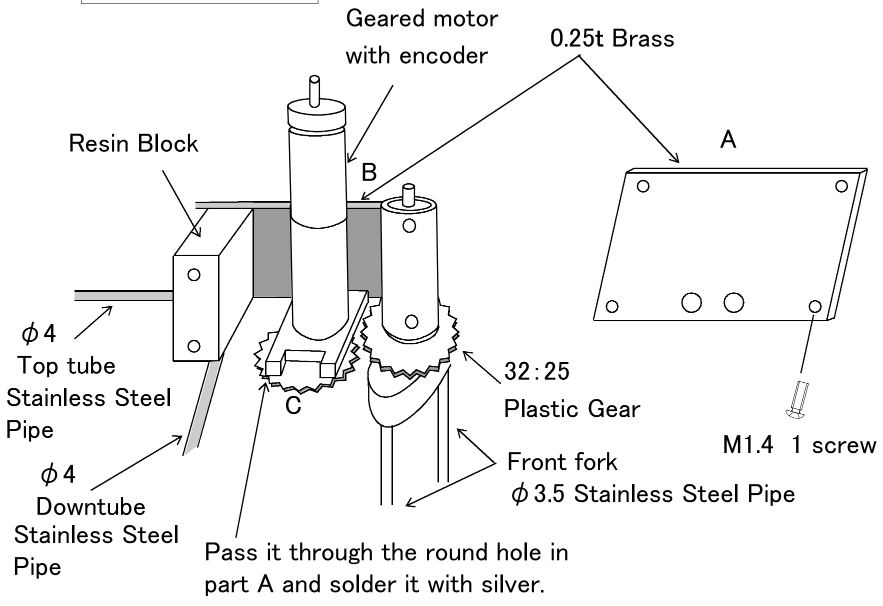


Photo 7.12

The control board under development and a dummy board used for measuring dimensions

Example of use

Gear mechanism



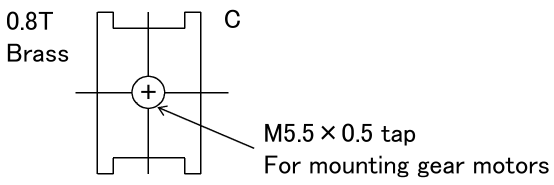


Figure 7.7 Usage example (1)

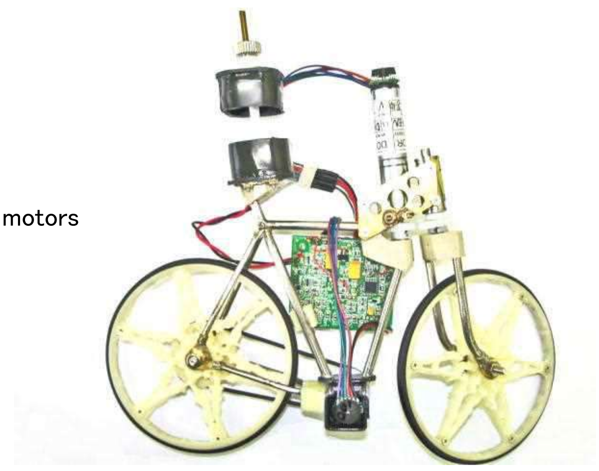


Photo 7.13
Front wheel fork shaft and gear mechanism of geared motor

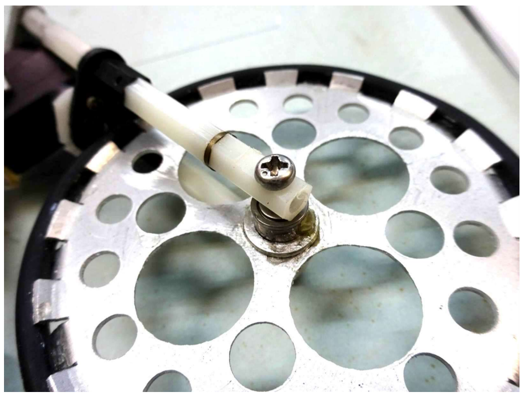
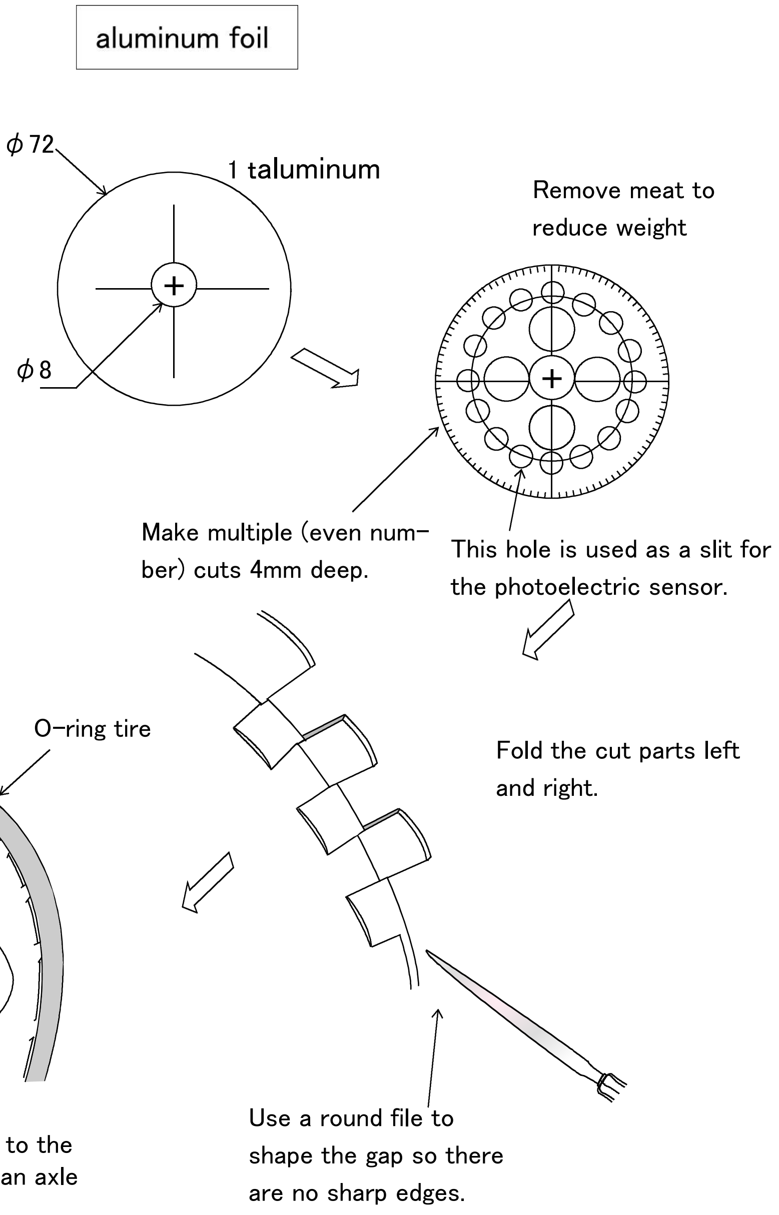
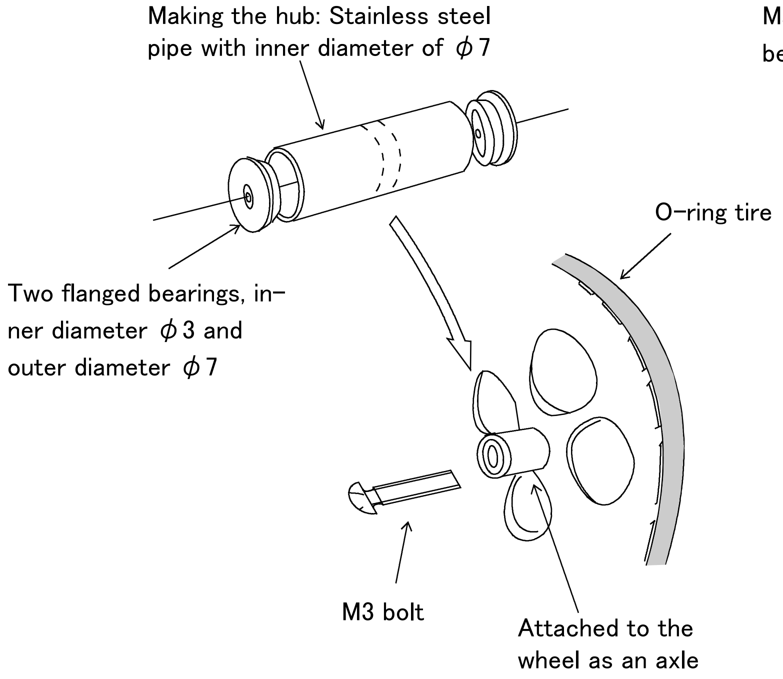
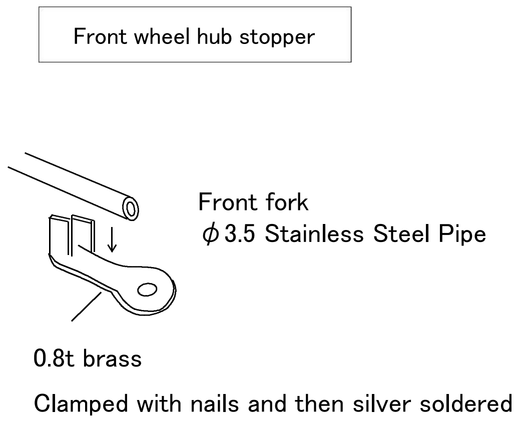


Photo 7.14
Example of making a wheel from aluminum plate


aluminum foil
φ72
1 taluminum
φ8
Remove meat to reduce weight
Make multiple (even number) cuts 4mm deep.
This hole is used as a slit for the photoelectric sensor.
Fold the cut parts left and right.
Use a round file to shape the gap so there are no sharp edges.


Making the hub: Stainless steel pipe with inner diameter of φ7
Two flanged bearings, inner diameter φ3 and outer diameter φ7
O-ring tire
M3 bolt
Attached to the wheel as an axle


Front wheel hub stopper
Front fork
φ3.5 Stainless Steel Pipe
0.8t brass
Clamped with nails and then silver soldered

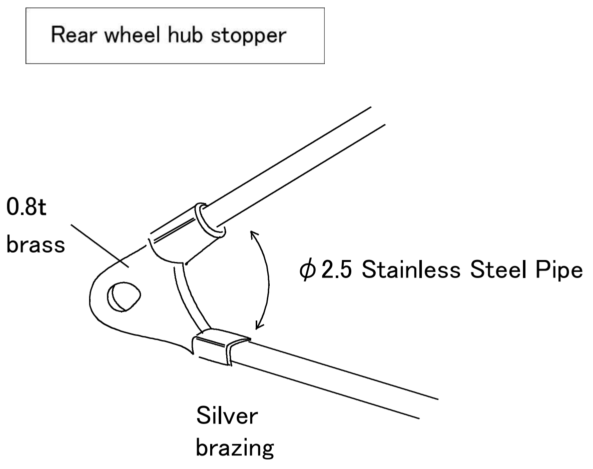


Figure 7.8 Usage example (2)

(5)Plastic Gear

Even if you use the reduction gear inside the servo, the torque transmission between the output shaft of the servo and the drive shaft of the handle, etc. will ultimately be done by gears. Normally, for this size, a plug gear with module 0.5 is used. If the module is 0.5, any manufacturer's gears can be combined. The combination used here is a spur gear, but there is also a worm gear. The reduction ratio can be 1:10 or more, but the transmission efficiency is poor and the torque generating axis of the worm and the mating axis are perpendicular, making the bearing structure difficult. This combination of spur gears also seems to be good for a reduction ratio of up to 1 :5 per pair. Since the final reduction ratio from the motor to the drive shaft is 1:64 to 1 :150, a combination of 3 to 4 stages can be used. If the reduction ratio of one pair is increased, the distance between the two parallel shafts will also increase, and one of the spur gears will become larger, making it difficult to work on the frame or wall where the bearings are located. Regarding the bearings for the gear shaft, if the plate thickness (in the case of brass) is 0.3 mm or more, it is fine to just drill a hole (φ 1 to 2 mm), but when it comes to the output shaft of the final stage, the shaft diameter becomes thicker and a force is applied in the thrust direction . If the plate thickness is 0.8 mm or more or is a thin plate, the shaft should be supported by a metal bearing or bearing.

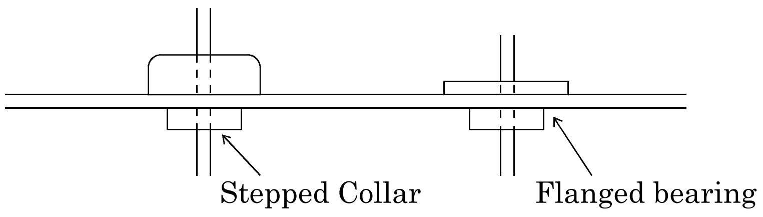


Figure 7.9

7.6 About radio controlled servos

This is a servo used in radio-controlled cars, which moves proportionally (proportional: the origin of the word "propo") in response to the tilt of the stick on the corresponding channel of the transmitter. There are various types of servos, from extremely small servos weighing 1.7g, to those used in large models such as industrial helicopters, high-torque servos with multiple rotations for sail winches on yachts, and servos specialized for robot joints. The servos used in bicycle development are small in size, have high torque and speed, use bearings on the output shaft, and have little radial backlash. The servos are as follows: Digital servo, metal gear, with bearings Model: CT12-DMG Input voltage: 4.8 -6.0V Speed: 0.12sec/60 ° Torque: 3.0kgm/4.8V Size: 12.0 x 23.2 x 24.8 (mm) Weight: 12.47 g Current consumption: 166mA

Photo 7.15
This is the servo I used the most this time.

When operating the steering wheel with the servo link mechanism

The front wheels can be driven by a servo link mechanism without using a spur gear. The range of motion and torque can be adjusted by changing the length of the link arm.

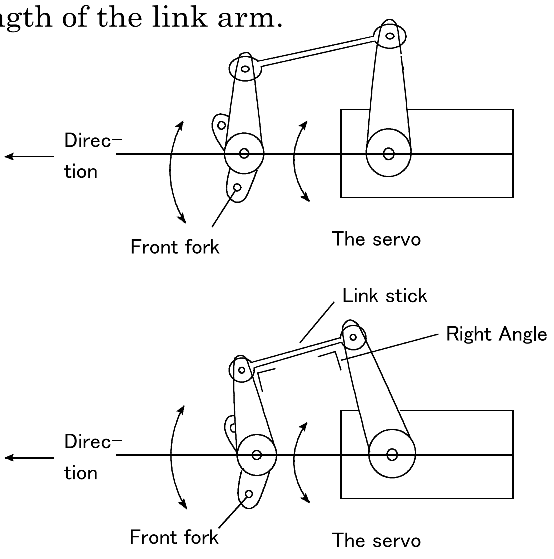


Figure 7.10

This installation allows the front fork to be wider than the servo's movable range, but the left side is wider than the right side, which causes imbalance. (This is called a differential linkage.)

When in neutral, each arm and link rod should be at a right angle.

This is how it is distributed left and right.

(In the world of radio-controlled cars, there are many places where differential linkages are required.)

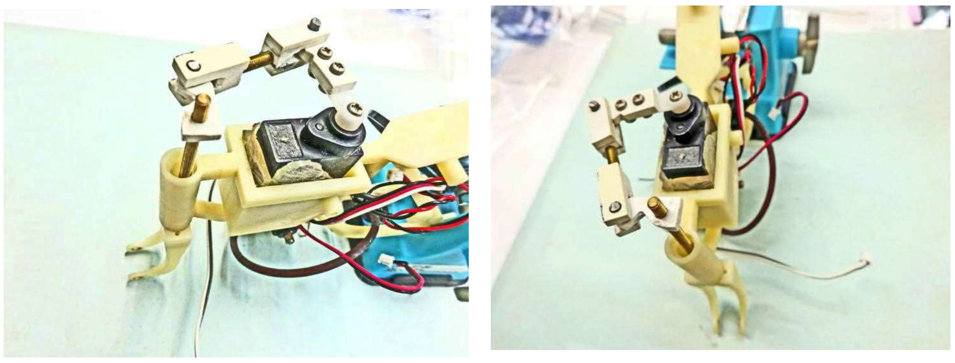


Photo 7.16
Example of front wheel drive by linkage

7.7 Servo Modification

This mechanism works by matching the feedback signal of the internal potentiometer to a pulse signal that moves between a narrow 1msec and a wide 2msec, with a pulse width of 1.5msec. It is possible to control it using the pulse signal as it is, but the movable range is ±30° (total 60°), so its use is limited.

In the end, only the gear mechanism part is left and used. Previously, the connections to the internal motor and potentiometer were soldered with lead wires, but the latest ones have the metal leads of the motor and potentiometer soldered directly to the servo board with protrusions, making them difficult to separate. Also, some stoppers are molded into the structure and cannot be removed.

When using the servo as a front wheel drive axle

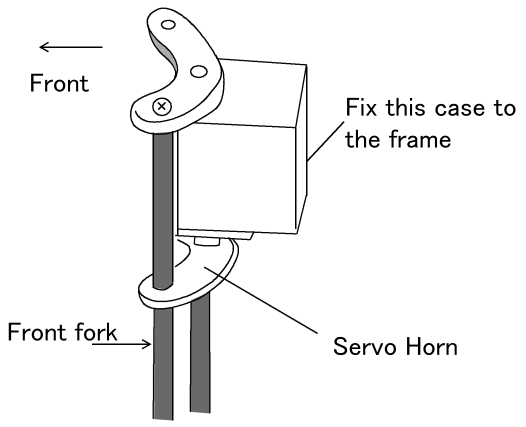
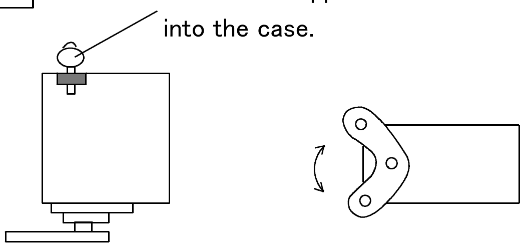


Figure 7.11



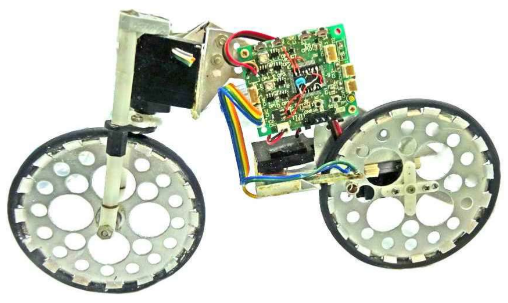


Photo 7.17
An example of using the servo shaft for the front wheel drive shaft. If the servo case is fixed to the chassis, it will be integrated with the frame.

When fixing a boss (gear, pulley) to a shaft with a screw

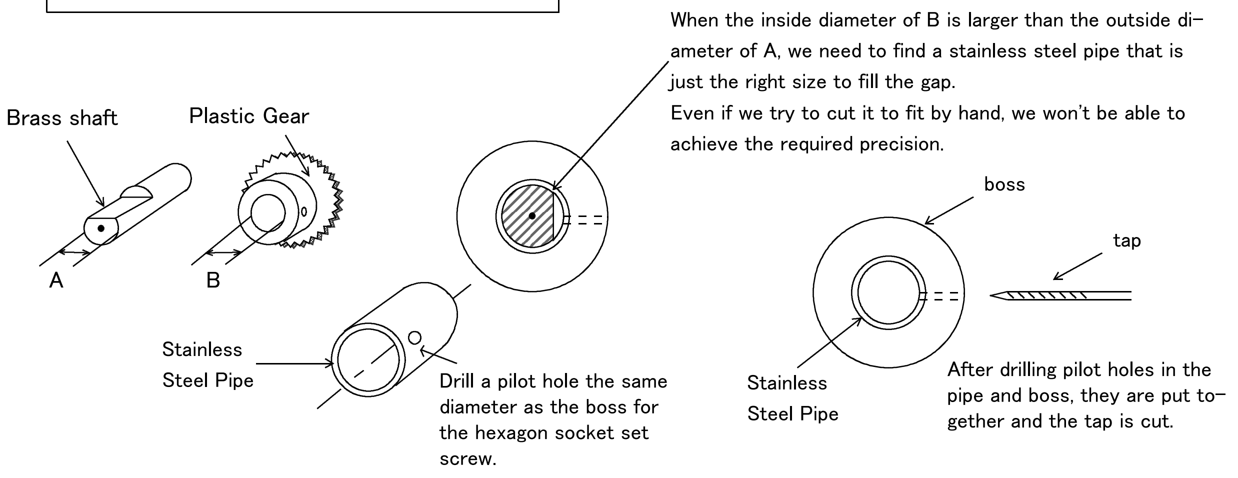

When the inside diameter of B is larger than the outside di–ameter of A, we need to find a stainless steel pipe that is just the right size to fill the gap.
Even if we try to cut it to fit by hand, we won't be able to achieve the required precision.
Brass shaft
Plastic Gear
A
B
boss
tap
Stainless Steel Pipe
Drill a pilot hole the same diameter as the boss for the hexagon socket set screw.
Stainless Steel Pipe
After drilling pilot holes in the pipe and boss, they are put to–gether and the tap is cut.

Figure 7.12

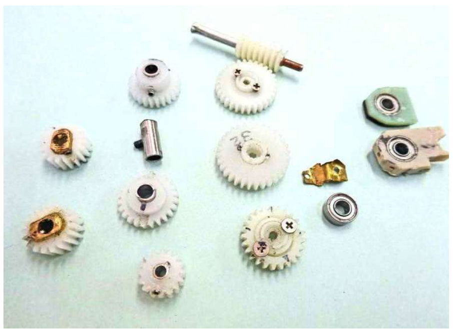


Photo 7.18
When fixing gears or pulleys to a shaft, the inner metal pipe is also pierced and tapped. This process requires high precision.

7.8 How to obtain parts and materials

Most industrial parts can be purchased online or at a hardware store.
A surprisingly good place to find them is the large fishing tackle stores. They stock interesting materials for people who want to make or repair their own tackle. For example, carbon rods, plastic tubes, stainless steel pipes, stainless steel wire, carbon sheets, resin adhesives, etc. Each item comes in a variety of sizes and can be purchased in small quantities. If you take a look at 100-yen stores, you'll find a lot of useful items.

7.9 Tools I wish I had

A bench lathe or drill press can be used to make a variety of processed products, but here we will consider how to do the work without them.
When using a drill to make holes in thin plates (around 0.2 mm thick) of resin or metal, the holes will tear and the holes will not be drilled cleanly.
There is a punch called a disc cutter. There are two thick metal plates with several holes ranging from φ4 to φ16, and the thin plate to be drilled is inserted into the gap of about 2 mm, and a pin that matches the diameter is driven in from above to make a round hole. The result is a clean finish, and the round piece that was inserted and punched can also be used.

A tapered reamer is used to enlarge the hole diameter of the shaft. If you enlarge it with a round file, the center will shift and the precision will not be achieved.
A metal bending tool (pocket bender: made by Engineer) can bend thin plates neatly at right angles. It can also be bent into U-shapes and stepped bends, and has a wide range of applications.

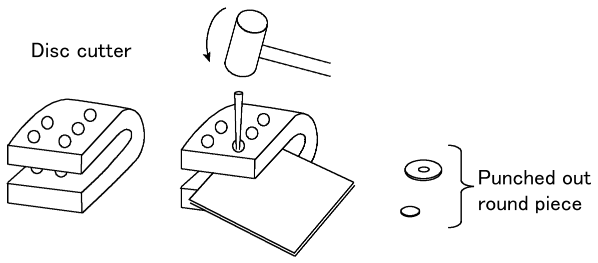


Figure 7.13

INDEX